# DEPARTMENT OF COMMUNICATION SYSTEMS

MASTER THESIS PROJECT

# IP Mouse

CRISTINA RIERA MENCIÓ

**Examiner and Academic Supervisor**
Prof. Mark T. Smith

TRITA-ICT-EX DEPARTMENT

Stockholm, Sweden 2010

# Abstract

Windowing systems [1] have been created to support a single user at the use of different applications. This Master Thesis consist of developing a new pointing device, from now called IP Mouse (IPM), that allows multiple cursors to interact with one application at the same time and supports the execution of a well-established applications in a multi-user context. That means that the system supports real time interaction in a shared display.

The IPM system is based on the PS/2 protocol, and the core of its hardware is a ColdFire microcontroller integrated in a Freescale board (M52233EVB) that controls and supports all the parts of the design and has the features to can send the data through Ethernet.

This work shows that the complete IPM system could be an effective support for multi-pointing interfaces on shared display systems.

# Acknowledgements

I would like to express sincere gratitude and love to my family for their endless support and inspiration during this period and my entire life.

I am very thankful to my supervisor and examiner, Professor Mark T. Smith, for giving me the opportunity to work on this amazing project and for his kind, advice and encouragement to help me complete this project.

I want to express my happiness to all my friends and colleagues who supported me during this year: Ana Santamaría, Elena Márquez, Fernando Guillén, Joaquín Juan, Luís Maqueda, Maria Kroich, María Gorrochategui, Oriol Boix, Ramon Claramunt, Víctor García. And I want to express my most sincere thanks to my friends Pablo Jorba and Fran Aleo for their help and knowledge with microcontrollers and programming.

I want to express my happiness also to all the friends that helped me during my university period: Yasmina, Miquel, Marc G., Aleix, Albert, Patri, Mireia, Nuria, Dani F., Lluís, Robert, Aïda, Lucía, Alberto, Txetxu, Dani O., and a long etcetera. Sorry I cannot mention you all!

I really had a rewarding experience at Wireless@KTH with the company of all the folks and colleagues.

Finally, I want to make a special mention to my grandfather, Llorenç, for his love during my entire life and for his endless support during my Erasmus period even on his last days.

# Table of Contents

**1 INTRODUCTION ...... 1**

**2 BACKGROUND ...... 3**

2.1 MOUSE HISTORY AND EVOLUTION ...... 3

2.2 MOUSE TECHNOLOGIES ...... 6

*2.2.1 First Mechanical Mouse ...... 6*

*2.2.2 Track ball mechanical mouse ...... 6*

*2.2.3 Optical Mouse ...... 7*

*2.2.4 Laser Mouse ...... 7*

*2.2.5 Wireless Mouse ...... 8*

2.3 PREVIOUS WORK ...... 9

2.4 WHY IPM? ...... 13

2.5 IPM COMPLETE SYSTEM OVERVIEW ...... 14

**3 IPM SYSTEM ARCHITECTURE ...... 15**

3.1 M52233EVB ...... 15

3.2 DESIGN OF THE PS/2 INTERFACE BOARD ...... 16

*3.2.1 Design of the PS/2 Hardware Interface ...... 16*

*3.2.2 Main Blocks ...... 17*

*3.2.3 Design of the schematic ...... 26*

*3.2.4 Design of the PCB ...... 27*

*3.2.5 Building the board ...... 28*

**4 IMPLEMENTATION ...... 31**

4.1 HUMAN INTERFACE DEVICES (HID) ...... 31

*4.1.1 The PS/2 Interface ...... 32*

*4.1.2 PS/2 General information ...... 33*

*4.1.3 PS/2 Communication Protocol ...... 34*

4.2 WINDOWING SYSTEMS ...... 40

*4.2.1 What is X-Windowing? ...... 40*

*4.2.2 The X server* ........ *42*
*4.2.3 X-Window System Core Protocol* ........ *42*
*4.2.4 Windowing hierarchy* ........ *44*
*4.2.5 Events* ........ *45*
4.3 NETWORK PACKETS ........ 45
4.4 CONNECTING THE BOARD ........ 47
4.5 PROGRAMMING THE BOARD ........ 49
*4.5.1 Testing the board* ........ *49*
*4.5.2 Interruptions Test Program* ........ *52*
*4.5.3 Final program* ........ *55*
*4.5.4 Implementation of the IP Mouse* ........ *71*
**5 DATA, TESTS AND EVALUATION ........ 73**
5.1 IPM INTERFACE ........ 73
5.2 TIMING AND FREQUENCY SPECIFICATIONS ........ 84
5.3 SENDING PACKETS INTERFACE ........ 85
5.4 IPM LIMITATIONS AND COSTS ........ 88
**6 CONCLUSION ........ 90**
**7 FUTURE WORK ........ 91**
**8 REFERENCES ........ 93**
**APPENDIX 1 ........ 102**
**APPENDIX 2 ........** ERROR! BOOKMARK NOT DEFINED.

# List of Figures

Figure 1. Multi-user application. Multiple mice on a windowing system........................ 2
Figure 2. IPM Complete System graphic ........................................................................ 14
Figure 3. Main blocks of the PS/2 interface hardware .................................................... 17
Figure 4. TPS61072’s Application Circuit for Adjustable Output Voltage ................... 18
Figure 5.  Simplified Outline of the BC546 transistor .................................................... 21
Figure 6. Open Collector circuit for the PS/2 mouse ...................................................... 22
Figure 7. IPM schematic design ...................................................................................... 26
Figure 8. IPM’s PCB design............................................................................................ 28
Figure 9. Front side of the PCB........................................................................................ 29
Figure 10. Back side of the PCB ...................................................................................... 29
Figure 11. Front side of the final PCB ............................................................................. 29
Figure 12. Back side of the final PCB.............................................................................. 29
Figure 13. Complete system from the top ........................................................................ 30
Figure 14. Profile of the complete system........................................................................ 30
Figure 15. 6-pin Mini-DIN (PS/2) connectors ................................................................. 33
Figure 16. Device to Host data packet and timing diagram ............................................ 35
Figure 17. Host to Device data packet and timing diagram ............................................ 37
Figure 18. X-Windowing client/server model.................................................................. 41
Figure 19. Communication interaction between a client and a server............................ 43
Figure 20. X-Windows hierarchy ..................................................................................... 44
Figure 22. TCP/IP Stack Overview .................................................................................. 46
Figure 23. M52233DEMO board ..................................................................................... 47
Figure 24. Hyper Terminal log window ........................................................................... 48
Figure 25. Mouse State Diagram....................................................................................... 56
Figure 26. SendData Flow Chart....................................................................................... 60
Figure 27. Sending and Reading data from a mouse....................................................... 63
Figure 28. Flow chart of readData.................................................................................... 63
Figure 29. flow chart for waiting state ............................................................................. 64

Figure 30. Flow Chart of D0 state ........................................................................ 66
Figure 31. Three bytes sent from the mouse when Left Button is pressed..................... 75
Figure 32. Left Button. First Byte ........................................................................ 75
Figure 33. Left Button. Second Byte..................................................................... 75
Figure 34. Left Button. Third Byte........................................................................ 75
Figure 35. Three bytes sent from the mouse when right button is pressed .................... 77
Figure 36. Right Button. First Byte ...................................................................... 77
Figure 37. Right Button. Second Byte.................................................................... 77
Figure 38. Right Button. Third Byte ..................................................................... 77
Figure 39. 3 bytes sent from the mouse when moved towards the X direction.............. 79
Figure 40. Right Movement First Byte.................................................................. 79
Figure 41. Right Movement Second Byte .............................................................. 79
Figure 42. Right Movement Third Byte ................................................................ 79
Figure 43. 3 bytes sent from the mouse when movement to the right has occurred ...... 80
Figure 44. Right Movement. First Byte................................................................. 80
Figure 45. Right Movement. Second Byte ............................................................ 80
Figure 46. Right Movement. Third Byte ............................................................... 80
Figure 47. Up movement. 3 bytes sent from the mouse ............................................ 81
Figure 48. Up movement. First Byte ................................................................... 82
Figure 49. Up movement. Second Byte................................................................. 82
Figure 50. Up movement. Third Byte................................................................... 82
Figure 51. Down movement. 3 bytes sent from the mouse ........................................ 83
Figure 52. Down movement. First Byte ............................................................... 83
Figure 53. Down movement. Second Byte............................................................. 83
Figure 54. Down movement. Third Byte............................................................... 83
Figure 55. Timing of a byte sent from IPM............................................................ 84
Figure 56.Timing between packets sent from IPM ................................................. 84
Figure 57. WireShark screen capture for left button ............................................... 85
Figure 58. WireShark screen capture for the right button ......................................... 86
Figure 59. Screen capture for mouse movement ..................................................... 87

# List of Tables

Table 1. Values of the TPS61072 application circuit components................................ 20
Table 2. NPN transistor Pin description.......................................................................... 21
Table 3. BC546 electrical characateristics ...................................................................... 21
Table 4 Electric features of MAX3224E.......................................................................... 25
Table 5. Movement Data Packet ...................................................................................... 32
Table 6. PS/2 protocol. Bus states.................................................................................... 34
Table 7. Movement Data Packet ...................................................................................... 73
Table 8. Left button expected vs. obtained values .......................................................... 74
Table 9. Right button expected vs. obtained values ........................................................ 76
Table 10. X movement expected vs. obtained values...................................................... 78
Table 11. Left movement (negative) expected vs. obtained values................................ 79
Table 12. Up movement expected vs. obtained values.................................................... 81
Table 13. Up movement expected vs. obtained values.................................................... 82
Table 14. Overview of the prices of the different components used and final cost of the IPM prototype............................................................................................................. 89

# List of Abbreviations

| | |
|---|---|
| **2D:** | Two-Dimension |
| **3D:** | Three-Dimension |
| **AC:** | Alternating Current |
| **ACK:** | Acknowledgement |
| **ADC:** | Analog to Digital Converter |
| **AN:** | Analogic |
| **API:** | Application Programming Interface |
| **ARP:** | Address Resolution Protocol |
| **AWG:** | American *Wire* Gauge |
| **BDM:** | Background Debug Module |
| **BIOS:** | Basic Input-Output System |
| **BSD:** | Berkeley Software Distribution |
| **CAD:** | Computer Aided Design |
| **CAN:** | Controller Area Network |
| **CD:** | Compact Disk |
| **CMOS:** | Complementary Metal Oxide Semiconductor |
| **COM:** | COMmunication |
| **CPN:** | Coloured Petri Net |
| **CPU:** | Central Processing Unit |
| **CRC:** | Cyclic Redundancy Check |
| **CW:** | Code Warrior |
| **DB-9:** | D-Shape Shell Body 9-pins |
| **DC:** | Direct Current |
| **DDRx:** | Data Direction Register |
| **DHCP:** | Dynamic Host Configuration Protocol |

| | |
|---|---|
| **DIN:** | *Deutsches Institut für Normung* (German Standardization Organization) |
| **DMA:** | Direct Memory Access |
| **DNS:** | Domain Name System |
| **DOS:** | Disk Operating System |
| **DSP:** | Digital Signal Processor |
| **EMAC:** | Ethernet Media Access Controller |
| **EMI:** | Electromagnetic Interferences |
| **EPORT:** | Edge Port |
| **ESD:** | Electrostatic Discharge |
| **ESR:** | Equivalent Series Resistance |
| **FB:** | FeedBack |
| **FEC:** | Fast Ethernet Controller |
| **GND:** | Ground |
| **GPIO:** | General Purpose Input/Output |
| **GPT:** | General Purpose Timer |
| **GUI:** | Graphic Unit Interface |
| **GWWS:** | Groupware Windowing System |
| **HCT:** | High-speed CMOS [logic] with TTT-compatible [logic] levels (IC, MOS) |
| **HID:** | Human Interface Device |
| **HW:** | HardWare |
| **I/O:** | Input/Output |
| **IBM:** | International Business Machine |
| **IC:** | Integrated Circuit |
| **ICMP:** | Internet Control Message Protocol |
| **ID:** | Identity |
| **IEC:** | *International Electro-technical Commission* |

| | |
|---|---|
| **IIC:** | Also I2C. Inter-Integrated Circuit |
| **IP:** | Internet Protocol |
| **IPM:** | Internet Protocol Mouse |
| **IRQ:** | Interrupt Request |
| **JTAG:** | Joint Test Action Group |
| **LED:** | Light Emitting Diode |
| **LSB:** | Least Significative Bit |
| **MAC:** | Macintosh |
| **MIPS:** | Mega Instruction per Second |
| **MIT:** | Massachusetts Institute of Technology |
| **MMM:** | Multi-Device, Multi-User, Multi-Editor |
| **MPWM:** | Multi-Pointer Window Manager |
| **MPX:** | Multi-Pointer X Server |
| **MSB**: | Most Significative Bit |
| **NACK:** | Negative Acknowledgement |
| **NPN:** | Negative-Positive-Negative |
| **OS:** | Operative System |
| **PC:** | Personal Computer |
| **PCB:** | Printed Circuit Board |
| **PDA:** | Personal Digital Assistant |
| **PIT:** | Programmable Interrupt Controller |
| **PLL:** | Phase Lock Loop |
| **PS/2:** | Personal System/2 (Protocol Serial 2) |
| **PTM:** | Point To Multipoint |
| **PTMP:** | Point To Multipoint Protocol |
| **PWM:** | Pulse-Width Modulation |

| | |
|---|---|
| **QSPI:** | Queued Serial Peripheral Interface |
| | |
| **RAM:** | Random Access Memory |
| **RF:** | Radio-Frequency |
| **RJ-45:** | Registered Jack 45 |
| **ROM:** | Read Only Memory |
| | |
| **RS-232:** | Recommended Standard 232 |
| **RTC:** | Real Time Controller |
| | |
| **SCTP:** | Stream Control Transmission Protocol |
| **SDG:** | Single Display Groupware |
| **SDGT:** | Single Display Groupware Toolkit |
| **SRAM:** | Static Random Access Memory |
| **SOTn:** | |
| **SSOP:** | Shrink Small Outline Package |
| **TCP/IP:** | Transfer Control Protocol /Internet Protocol |
| **TIDL:** | Tool Integration Description Language |
| | |
| **UART:** | Universal Asynchronous Receiver/Transmitter |
| **UDP:** | User Datagram Protocol |
| **URL:** | Uniform Resource Locator |
| **USA:** | United States of America |
| **USB:** | Universal Serial Bus |
| | |
| **XMP:** | Extensible Metadata Platform |
| **XMP:** | X Multi Pointer |
| **XP:** | eXPerience |
| | |
| **µC:** | microController |

# 1 Introduction

Nowadays everybody has a computer and has worked with it. Users are used to having their computer screen and some peripheral devices such as a keyboard, a joy-stick or a mouse. People with a laptop can either choose if they want to use the touchpad, a peripheral mouse, or both. At present, if more than a mouse is plugged in a computer, there is always only one pointer.

Windowing systems [1] have been designed to support multiple applications at the same display just by dividing the screen, so one user can run multiple applications. These systems are scalable, intelligent and able to control each part of the screen. Even so, users of windowing systems can only interact with one of the applications at the same time, because they still have only one single cursor.

Although windowing systems are a great solution to run multiple applications at the same time, users can wonder how it would be if besides of having multiple windows, multiple mice were able to be connected to one machine and work simultaneously without interfere with each other. That would be the perfect environment for applications that use more than a window (i.e. graphic applications, that use to have several windows to show tool bars such as layers or colours) or to run actively different applications simultaneously.

By contrast, instead of having one single user and many applications, a very common situation is to find many people in front of a single display (i.e. in a conference, meeting or teamwork). In that situation, multi-pointing devices could be a very useful tool to get active collaboration and a better approach from the users with the application running.

According to this, the idea of having a multi-pointing device, which could be built-in to any wearable electronic device such as a mobile phone or a PDA, could be very useful in real time collaboration with one or more applications in multi-user context situations.

This project consists of designing and building a virtual mouse device with Ethernet features capable of supporting multiple cursors on the same display. The virtual mouse motion is based on the PS/2 protocol [2] and its hardware is built around a 68k/ColdFire microprocessor [3] attached on a Freescale Evaluation Board [4]

The virtual mouse, from now called IP Mouse (IPM), must be able to perform the basic 2D-motion movements of a conventional mouse and also must have the features to send its data to the system required through Ethernet.

This work presents the research, design, building and implementation of the IPM device, which is intended to be an effective support for multi-pointing interfaces on shared display systems, with multiple applications in a multi-user context.

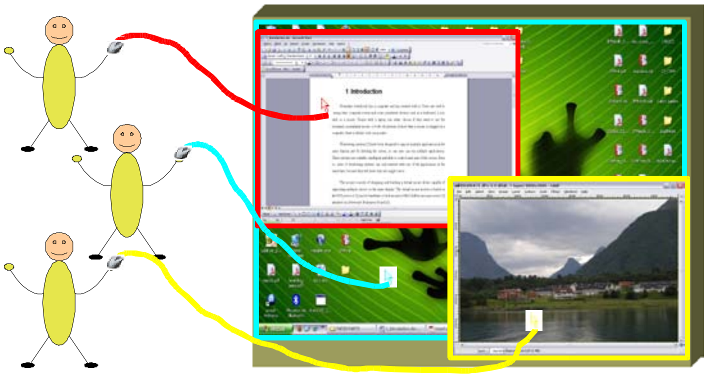


Figure 1. Multi-user application. Multiple mice on a windowing system.

# 2 Background

Something new needs to be created. Therefore, the review of existing technology and knowledge related to this area could be relevant to understand and solve the problem described on chapter 1. It is important to know what devices exist, what kind of device needs to be created and how should it work. A good start point is to start researching about the existent devices and the evolution they suffered across the years in terms of technology.

## 2.1 Mouse History and Evolution

A computer mouse is a very common pointing device. Its aim is to detect two dimensional motions relative to the surface where it is supported and translate this movement into the motion of a cursor on a display.

The appearance of the first mouse redefined the way of interaction with the computers. Since then, its purpose is the same but its shape and technology have evolved and improved in terms of position accuracy, material and technology. Hereafter, some history of the mice from its beginning until the present days is presented.

The first mouse prototype was invented in 1963 by Douglas Engelbart. Engelbart was an American inventor and is known for the invention of the mouse and for being pioneer in human-computer interaction.

Engelbart's prototype was patented in 1970 under the title "X-Y Position Indicator for a Display System" [5]. The mechanism of this device consisted of two perpendicular toothed wheels, allocated in a way that the rotation of each wheel was transferred into motion along one of the respective axes in a plane.

It was 8 years later, in 1971, when Bill English, engineer and Engelbart's work collaborator, improved the first mouse prototype by replacing the toothed wheels with a metal ball.

The track ball could rotate to any direction and it was pressed against metallic rollers for tracking the movement, so the mouse could drive a pointer around the screen with ease.

Some years later, the professor Jean-Daniel Nicoud improved the mouse model incorporating a hard rubber ball and three buttons. Nicoud's design lasted until 1985 when René Sommer added a microprocessor to the mouse.

Computer mice appeared on the market in the early 1980s. By those days, to can use a mouse, an additional expansion board/card was required, so mice didn't became popular until the late 1980s when many of IBM [6] began to sell computers with a serial port build-in and Apple Macintosh [7] began to sell their computer systems with a mouse included. Most mice had either one or two buttons and were connected to the computer through a serial connector.

It was in the early 1990s when mice increased considerably its popularity, with Microsoft [8] claiming mouse's benefits with the Windows 3.1 package, which had just been released in 1992.

IBM was the first company to introduce the PS/2 connector for mice and those connectors started gaining popularity over the serial port models. The popularity was so big that in the late 1990s, many computers came with another PS/2 port built-in specifically for the mouse.

In the late 1990s, the mouse model changed once again with the apparition of the scroll-wheel mouse and USB connectors started being used. USB became more widespread and gained popularity over PS/2 mice.

Mice technology improved quickly during the 1990s and towards the beginning of 2000s appeared the first optical mice. It was developed by Agilent Technologies [9] but many peripheral manufacturers adopted the model quickly.

At the beginning of the 21st century, technology was evolving fast and wireless communications were very popular, so mice started using wireless technologies. Most popular are RadioFrequency (RF) and Bluetooth.

Moreover, a laser design based on the optical mice was developed and appeared on the market. Although laser mice have much better tracking solutions, it is not well extended yet, because of the elevate price over the other existent devices.

At present, new models of mice have appeared, featuring three dimensional motion and gyroscopic solutions. An example of a 3D pointing device is the Wii Remote, which is a motion-sensing device. It can detect its spatial position by comparing the distance and position of the lights from the infrared sensors.

Overall, computer mice have evolved since its beginning. Different ports have been used, the mouse housing material and amount of buttons has changed and it is already possible to include them with new systems and technologies, such as electronic wearable devices (i.e. mobile phones or PDAs).

## 2.2 Mouse Technologies

Despite that over the years the mice’s principle remains the same: to translate motion from a physical surface onto motion of a cursor on a display, its mechanism and technologies have evolved. Different mice models functioning are explained on the next subsections.

### 2.2.1 First Mechanical Mouse

The first pointing device prototype consisted of a small wood housing, two wheels and a bearing ball for contacting the surface on which it was supported. The two wheels were orthogonal to each other and there was a potentiometer attached to each wheel in a way that when the device was moved, the two wheels rotated changing this way the resistance of the potentiometer.

Electrical cables connected the potentiometers to a computer which continuously monitored the device’s position. The computer displayed then a cursor, which moved accordingly with the movement of the device.

### 2.2.2 Track ball mechanical mouse

This model was based on the first prototype but its mechanism was improved. It was so successful that despite advances in material and technology, a basic ball mouse can still be easily found today.

A track ball mouse consists of one ball and two rollers oriented orthogonal to one another so they can detect movement in the X and Y direction respectively. As the mouse moves, the ball inside the mouse touches the supporting surface and roll. Consequently, one or both of these rollers rotate as well. The rollers each connect a bar, and the bar spins a disk with holes in it.

There is also an infrared LED and an infrared sensor, one on each side of the disk. As the mouse move, the ball rolls and so does the bar and the disk. The holes of the disk break the beam of light coming from the LED so that the infrared sensor sees pulses of light.

The rate of the pulsing is directly related to the speed of the mouse and the distance it travels. A microprocessor reads the pulses from the infrared sensors and turns them into binary data that the computer can understand.

### 2.2.3 Optical Mouse

Optical mice mechanism consists of a small, red light-emitting diode (LED) and a complimentary metal-oxide semiconductor (CMOS) sensor. The LED emits light onto the surface where the mouse rests and this light bounces off the surface onto the CMOS sensor.

The CMOS sensor sends each image to a digital signal processor (DSP) for analysis. Then the DSP detects patterns in the thousands of images sent and compares the new pattern with the previous image. Based on the change in patterns over a sequence of images, the DSP determines how far and at what speed the mouse has moved.

The DSP then sends the corresponding coordinates to the computer, which moves the cursor on the screen accordingly with the received coordinates from the mouse. This happens hundreds of times each second, making the cursor appear to move very smoothly.

The benefits of optical mice over track ball mice are that they have more accuracy, don’t require any maintenance and last longer due to fewer moving parts. Moreover, they don’t require a special surface, such as a mouse pad.

### 2.2.4 Laser Mouse

A laser mouse works identically as an optical mouse but uses an infrared laser diode instead of a LED to illuminate the surface below their sensor. The use of the laser diode increases about a 20 times the sensitivity to the surface, so the laser mouse tracking is much better, has better response times and can be used on even more surfaces compared to the ones using LED technology.

### 2.2.5 Wireless Mouse

Wireless mice mostly use radio frequency (RF) technology to send information to a computer. The most common type of RF used are the 802.11b or 802.11g. These standards operate at 2.4 GHz and guarantee quick data transfer speeds, usually either 11Mbps or 56 Mbps.

RF devices require two main components: a transmitter and a receiver. In order to avoid other RF devices interferences, the transmitter (usually included inside the mouse) and its receiver must be paired. Two devices are paired if they operate at the same frequency on the same channel using a common identification code.

Some benefits of RF devices over an infrared technology are that they do not need a clear line of sight between the transmitter and the receiver, RF components are inexpensive and RF transmitters require low power and can run on small batteries.

Another wireless technology emerging for mice is Bluetooth. Bluetooth mice operate on the same frequency 2.4 GHz, however, it also uses software called adaptive frequency hopping to choose frequencies that have no or little interference. Bluetooth also has a range of 10 meters.

## *2.3 Previous Work*

As it has been written at the beginning of this chapter, knowing about what others have done previously is a good way to start working yourself. Different works relating to the areas of mice, multi-user application and multi-pointer have been looked at and the ones considered more relevant are briefly resumed on the next lines.

During the research, this reference showed up unexpectedly and I considered it had to be mentioned:

*"...Using two hands is the natural mode people use to interact with the physical World*. Stewart et al. mentioned that at 1998. He meant that users do not like to share input devices when working collaboratively..."

At 1999, Stewart examined the effectiveness of **Single Display Groupware (SDG)** [10] with the help of a group of elementary school children. SDG stands for applications that run on a single computer display but allow multiple users to interact with one application simultaneously. Steward wanted to design and build a prototype SDG system called Sushy, consisting in an authoring tool for interactive multimedia stories. SDG applications and toolkits require the support for multiple input devices.

At 1991, Bier worked in the **MMM (Multi-Device, Multi-User, Multi-Editor)** [11] project. MMM is one of the earliest implementations of SDG. MMM enabled multiple users, each with an independent input device, to interact with multiple editors on the same computer display.

In 2004, **Wallace** et al. [12] developed a multi-cursor window manager where users could connect to a shared display over a network. The Window Manager assigned one different pointer for each connection. There was no support for multiple input devices, so the cursor had to be time-sliced and moved to the position of virtual cursor when an event occurred. This worked for single-time events such as mouse moves and button events but didn't work for events that required state information of the pointer.

The system worked as a windowing system and its behaviour was unpredictable when multiple input devices were used.

At 2006, Peter Hutterer implemented **the TIDL** (Transparent Input Device Layer) [13]. TIDL is a toolkit that permits to use an arbitrary number of independent input devices based in Java Swing API applications. The input devices connect to the system over a network and can be distributed amongst any number of host computers. This allows multiple users to be in front of the same host and collaborate on the application displayed.

Later on, Peter Hutterer and Bruce H. investigated how to integrate **groupware support for Single Display Groupware (SDG)** [14]. A GroupWare Windowing System (GWWS) combines the traditional single-user single-input event and newly multi-input desktop environments.

They presented the Multi-Pointer X Server (MPX) [15], which together with the Multi-Pointer Window Manager (MPWM) was the first GWWS that fully supported SDG. MPX and MPWM support an arbitrary number of cursors, sophisticated floor control, which means that resources could be utilized and shared without access problems, and per-window annotation overlay. They also implemented the Device Shuffler, a system that was located between the operating system and the device driver whose function was to couple input devices from any computer.

The physical connection point of a device was transparent to both the windowing system and the application. The system altogether supported multi-user collaboration on shared screens.

With the MPX, they enhanced the X Window System to support up to 255 independent mouse cursors. The cursors can operate in multiple applications independently. The MPWM actively supports multiple input devices for window operations.

There is also another project, named **Multiple Mice for computers in education in developing countries** [16] in which Udai Singh Pawar, Joyojeet Pal and Kentaro Toyama studied the behaviour of children in front of a computer of schools in developing countries. They noticed that, due to economic constraints, student-to-computer ratios ranged up to ten-to-one. Previous research in learning showed that involvement is an important factor of effective learning. Thus, they observed that active participant students were likely to be receiving more educational value than the ones without access to the mouse.

In consequence, they proposed, implemented and tested a technical solution to this problem by providing one mouse with its correspondent cursor to each child. This way, each child could have active interaction with the software. They implemented a Single Display Groupware (SDG) in developed countries' classroom.

Based on previous work and tests, they concluded the project claiming that in an educational context, one-to-many ratio of PC's to students is desirable, provided that every student has some control over what is happening on the screen.

At 2009, Tsutomu Kawai, Joutarou Akiyama and Minoru Okada proposed the **Point-to-multipoint communication protocol (PTMP) [17].** The system is intended to distribute and display identical data on many workstations simultaneously on the X window system with the stream-type data communication.

PTMP is intended to communicate with high performance and reliability on requiring low bandwidth. In order to implement that, PTM protocol combines the efficiency of broadcast, which is the easiest way to send information to multiple workstations, and the reliability of TCP/IP, which helps controlling the right reception of the data.

PTMP adopts the selective-repeat algorithm, which consists on sending a negative acknowledgement (NACK) only if the packet is erroneously received and suppressing the acknowledgements (ACKs) if the packets are received correctly.

PTM protocol sends basic data using broadcast protocol and uses UDP point to point protocol with some reliability added for the messages control. After analysis and tests they observed that PTMP was better than TCP/IP as the amount of clients increases. Even so, the protocol needs to be improved and better evaluated in terms of logical limitation of clients and completeness.

A **Keyboard Video Mouse switch (KVM)** [18] is a hardware device that allows controlling many computers or servers with only a monitor, a keyboard and a mouse. Usually KVM switches can control up to 64 computers at once.

At present, there are also devices able to share USB devices and speakers with multiple computers. By contrast with the usual KVM switches, some devices can function in reverse configuration, that is, one single PC can be connected to many monitors, keyboards and mice. This configuration is useful when a user wants to access a single computer from different locations.

KVM are very useful to control many computers which must be accessed independently and they can be accessed by hot keys or quick commands that can be used to access a specific computer or server fast and easily.

There are also KVM switches that allow the manageability of PC or servers through the TCP/IP connection. It is possible to access to a computer through internet, being in another location but feeling as if he was sitting in front of it.

**Mobile Air Mouse** [19] is a brand new application developed for RPA Tech [20]. That application instantly transforms an i-Phone or i-Pod Touch into a wireless mouse for any MAC or PC. It can operate into three different modes: air Mouse, track pad and wireless remote control for the computer.

The air mouse uses the built in accelerometer to translate hand motions into mouse movements on the screen. Mobile air mouse can also operate as a track pad, allowing controlling a computer with a single finger. Track pad mode is easier to use, because it is like a usual track pad of a laptop.

Both modes have the screen divided into two different parts. The top half of the screen is reserved for the mouse options, and the bottom half part is reserved for the usual i-phone keyboard with 3 keys added: Control, Alt and Command keys.

This application can also work as a remote control of the computer. Using an innovative application notification system, the i-Phone will always know what applications are running at any time and show the appropriate keys for that program, providing the user with a single screen for controlling all the media and web applications. It has no screen size limitation and works with multiple monitors.

## 2.4 Why IPM?

IPM is an innovative and interesting idea, because there have been many attempts but not many successes on this area. The purpose of IPM is similar to the goal pursued in many of the works presented before: to find out how to build and implement a multi-pointing device capable of being coupled transparently to any computer without interfere with other pointing devices that may be connected to the same machine, allowing this way multi-user collaboration.

Despite of the idea is similar, the methods and results used are different for each work. Nevertheless, the work of Hutterer and Bruce H., with the MPX system, is very relevant for this work because it shows that it is possible to do such a thing.

Even so, IPM device have another purpose, which is to send the device’s data to any computer through TCP/IP protocol. The device needs to be newly created and needs to fulfil some requirements. It has to be able to perform 2D-motion, and need to have Ethernet features.

Moreover it is desirable for the device to be flexible, as it may be included to any electronic device in a future. However, this is only the first prototype and implementation of IPM. Next section describes the functional overview of IPM device and its design and implementation.

## 2.5 IPM Complete System Overview

In this section the complete system is briefly explained and analyzed. The main parts of the system are shown on the figure below:

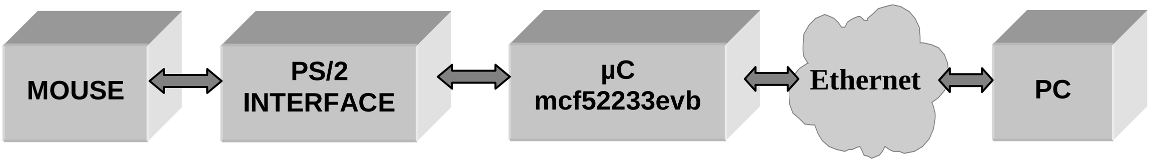


Figure 2. IPM Complete System graphic

The Mouse together with the microcontroller and the PS/2 interface constitute an independent pointer device that can be connected to any computer. The bus between them is bidirectional, which means that data can be transferred in both directions, from the PC to the device (i.e. when sending configuration commands or acknowledge bit) and from the device to the computer (i.e. when sending data information from the device).

Most of the operations are controlled by the microcontroller, which controls every process, movement or event occurred and received. The microcontroller controls all the hardware components such as the ports, connectors, Ethernet, etc. and provides the appropriate communication between the device and the host computer.

The computer acts like a server that establishes communication with the mouse and send the commands to set the correspondent configurations to the device. Afterwards, it stays waiting and listening to the correspondent port to receive the data transferred. Any external program previously run on the computer could take the data received and process it (i.e. print it on screen).

Subsequent chapters detail the hardware and software design and implementation of each part of the IPM system.

# 3 IPM System Architecture

This chapter details the different parts introduced on the overview of the IPM system. It breaks down into two main subchapters: M52233EVB, where the board features are explained, and PS/2 mouse interface hardware, where all the different blocks and components used on this part are described.

## *3.1 M52233EVB*

The DEMO52233 board is the main board used for this project. This board is an evaluation board for the MCF52233 ColdFire microcontroller [4]. The M52233EVB includes a DB9 serial port, USB (Background Debug Module) BDM port and Ethernet port. It is provided with a development kit, which includes a special edition of software support Code Warrior IDE [21] which supports application development and debug.

The Freescale board consists of many modules, timers, connectors and features, which can be useful for many applications and different projects, but the most relevant features for the development of the IP Mouse are:

M52233evb uses V2 ColdFire Core. The ColdFire V2 core is based on a memory-configurable hierarchical architecture that is 100 percent synthesizable, which means that it is conceived to be reused and integrated into custom designs easily. It has up to 46 Dhrystone 2.1 MIPS at 50MHz. A Dhrystone is a benchmark program that measures the system’s integer performance. Dhrystones per second is the metric used to measure the number of times the program can run in a second.

Another feature that will be used for the IP Mouse design is the Ethernet Media Access Controller Module (EMAC) and Hardware Divide, as the virtual device will need to send data through Ethernet.

Regarding the memory modules of the M52233EVB, it has 32 Kbytes of SRAM (Static Random Access Memory) and 256 Kbytes of flash memory. Moreover, once the data is saved, 10 years of data retention are guaranteed.

The Freescale board needs 3.3V of power supply and it can be powered up by the USB cable or by the 2.0 mm Barrel Power Input. The demo board also includes a USB, Serial, Ethernet cables and a 40-pin header connector, which gives user access to most of the I/O signals from the ColdFire microcontroller.

Freescale provides a free public source TCP/IP stack on their website. This TCP/IP stack is documented thoroughly in applications note AN3470, and the IPM TCP/IP stack is based on that application. Both files can be obtained in the Freescale official website [22].

## *3.2 Design of the PS/2 interface board*

The demo board have been introduced on the previous section, but as the aim of this project is to design a virtual mouse, there’s a need to design and build the hardware interface to connect the mouse with the Freescale board.

In order to implement the mouse interface, many interfaces could be used: serial, USB or PS/2. For this project the PS/2 interface, which will be further detailed on the Implementation section (chapter 4), has been chosen. Next subsections detail the design goals and the main blocks of the PS/2 interface board and the searching and choosing of the different components**.**

### 3.2.1 Design of the PS/2 Hardware Interface

Before designing the parts, some things need to be considered. It is important to know exactly what needs to be done and what features are required.

Based on the IPM system is using the Freescale DEMO board described at the previous section, which works at 3.3V, and knowing that there’s the need to include a mouse, which works at a voltage of 5V, there should be included some voltage regulator on the design. An elegant way to solve this problem is using a boost device.

Moreover, the system might need an extra buffer and the output ports: serial and PS/2. Some resistors, capacitors and inductors should also be used according to the design needs of the different parts.

### 3.2.2 Main Blocks

Having the problem statement and the requirements clear, one can proceed to the search of the right components for designing and building the system. This section presents the main blocks of the prototype design of the IPM's PS/2 interface hardware.

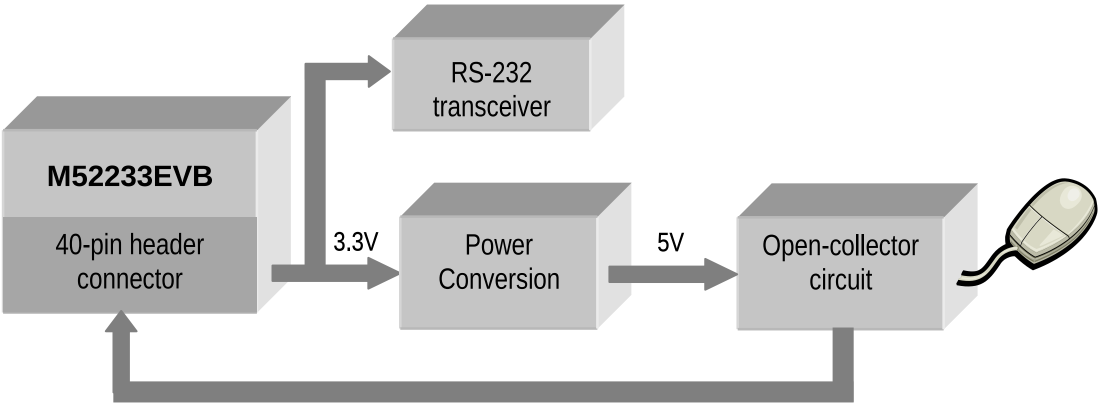


Figure 3. Main blocks of the PS/2 interface hardware

From the figure, the link between the demo board and the PS/2 interface is clear. The connection between the two boards is done through the 40-pin header connector from the demo board. The first pin of this connector is the power supply and it is 3.3V. The Power conversion block takes this voltage and converts it to a 5V voltage that will be used for interfacing the mouse with the open-collector circuit.

The output of the open-collector goes back to the 40-pin connector of the demo board, where the microprocessor will control and proceed with the operations. Moreover, there is an RS-232 transceiver included on the design, which includes a serial connector. Each block of the PS/2 interface is detailed on the next subsections.

### 3.2.2.1 Power subsystem

The demo board must be connected to the PS/2 interface. As have already been presented, the Demo board works at 3.3 V and the input of the PS/2 interface needs 5V.

To solve this little problem of power conversion, and after ask what devices were available, the **TPS61072** [23] device was selected. TPS61072 is a synchronous boost convertor from Texas Instruments [24].

The boost converter supports input voltages between 0.9 and 5.5 V. It is based on a fixed frequency of 600 kHz and synchronous rectifier to obtain maximum efficiency. The output voltage regulation is only maintained when the input voltage applied is lower than the programmed output voltage. Its maximum peak current is limited to 600mA.

The TPS61072 output voltage is programmed by an external resistor divider. The converter can be disabled in order to enlarge the battery life, and during shutdown, the load is completely disconnected from the battery to minimize the current drains. The device is packaged in a 6 pin SOT23 package. The circuit used to adjust the output voltage with this boost regulator is the following:

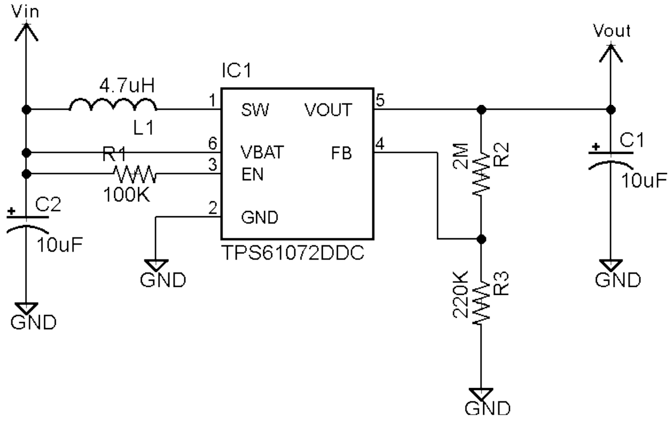


Figure 4. TPS61072’s Application Circuit for Adjustable Output Voltage

In the figure 4, $V_{in}$ is the input voltage and $V_{out}$ the output (or desired) voltage. The most important components in the figure are the inductor (L1), the boost regulator (IC1), the resistors at the output and the capacitors.

In order to calculate the right values for the components of that configuration, some electrical characteristics from the regulator device must be considered. Those can be found on the regulators' datasheet [23].

The most important characteristics are:

- The typical value of the voltage at the feedback pin (FB) is 500 mV.
- The maximum recommended value for the output voltage is 5.5V.
- The current through the resistive divider should be about 100 times greater than the current into the FB pin.
- The typical current into the FB pin is 0.01 µA.
- The voltage across R2 is typically 500 mV.

The feedback voltage is useful to fix the output voltage of the regulator. This voltage is present between the FB pin and GND (as shown in figure 4). The feedback voltage will be useful to calculate the values of the output resistors to achieve the desired output voltage.

Based on the recommendations from the datasheet, the value for R2 should be in the range of 200 kΩ. From that, the value of resistor R1, depending on the needed output voltage (Vo), can be calculated using the equation:

$$R1 = R2 \times \left( \frac{V_O}{V_{FB}} - 1 \right) = 220k\Omega \times \left( \frac{V_O}{500mV} - 1 \right) = 220k\Omega \times \left( \frac{5V}{500mV} - 1 \right) \approx 2M\Omega$$

Note that R2 has been taken 220 kΩ (in the range of 200 kΩ) and that the desired output voltage of the system is 5V.

As presented on figure 4, a boost converter normally requires two main passive components for storing energy during the conversion. A boost inductor and a storage capacitor at the output are required. Regarding the capacitor selection, 10 µF or higher input capacitor is recommended to improve transient behaviour of the regulator and electromagnetic interferences (EMI) behaviour of the total power supply circuit.

In the IPM design, datasheet recommendations of inductance and capacitors selection have been taken. Therefore, the inductance selected is, as in typical applications, of a value of 4.7µH and the capacitor values selected are 10 µF.

Taking all the considerations the final values obtained are:

| Component | Value |
|---|---|
| R1 | 100 kΩ |
| R2 | 2 MΩ |
| R3 | 220kΩ |
| L1 | 4.7 µA |
| C1 | 10 µF |
| C2 | 10 µF |

Table 1. Values of the TPS61072 application circuit components

From the datasheet, one can see that the feedback voltage is a pseudo constant voltage between the FB pin and Ground. This is useful to calculate the output voltage. The FB voltage forces a constant current of $I_{bias} = \frac{0.5}{R_2}$ that supposes a voltage drop of $V_{drop} = I_{bias} \times R_1$. Joining the two equations, the resultant output voltage can be calculated following the expression: $V_{out} = V_{FB} + V_{drop}$ .

Using the values obtained, IPM system have an $I_{bias}$ = 2.27µA. What makes the $V_{drop}$ = 4.54 V and an output voltage obtained of $V_{out}$ = 5.04V. The system accomplishes the requirements mentioned on page 19.

#### 3.2.2.2 Open-Collector Interface

In order to connect a PS/2 mouse to the Freescale board, it is required to build an open-collector circuit to make the interface between the two boards. For the design of the open-collector circuit, NPN transistors and pull-up resistors will be used. Next lines describe these components and its function.

### BC546 [25]: NPN general purpose transistors

The NPN transistors general purpose is switching and amplification. These transistors have low current (max. 100 mA) and low voltage (max. 65 V).

The transistor outline is:

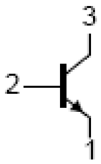


Figure 5. Simplified Outline of the BC546 transistor

| PIN | DESCRIPTION |
|---|---|
| 1 | Emitter |
| 2 | Base |
| 3 | Collector |

Table 2. NPN transistor Pin description

And its main characteristics are:

| Parameter | Description | Min Typ Max | Unit |
|---|---|---|---|
| $h_{FE}$ | DC current gain | 150 | |
| $V_{CE}$ | Collector-emitter voltage | 5 | V |
| $V_{BEsat}$ | Base-emitter saturation voltage | 700 | mV |
| $V_{BE}$ | Base-emitter voltage | 580 660 700 | mV |
| Vce | Collector-emitter | 100 | mV |

Table 3. BC546 electrical characteristics

### 74LCX86: Low Voltage Quad 2-Input XOR Gate with 5V tolerant Inputs.

The LCX86 [26] contains four two-input exclusive-OR gates. It is fabricated with advanced CMOS technology, which allows high speed operation and low power dissipation. The inputs tolerate voltages up to 7V allowing the interface of 5V systems to 3V systems.

The LCX86 are adequate for the mouse interface, because even if they have a 5V of input voltage, they have an output of 3.3 V at the highest. This property is useful to convert the 5V coming from the PS/2 mouse to the 3.3 V needed for the DEMO board.

Another feature that is good for the project is that this component output is low if the two inputs are equal, and the output is high if the two inputs are different. In other words, it can work both as a buffer and as an inversor.

The PS/2 interface schematic used for this part is shown below:

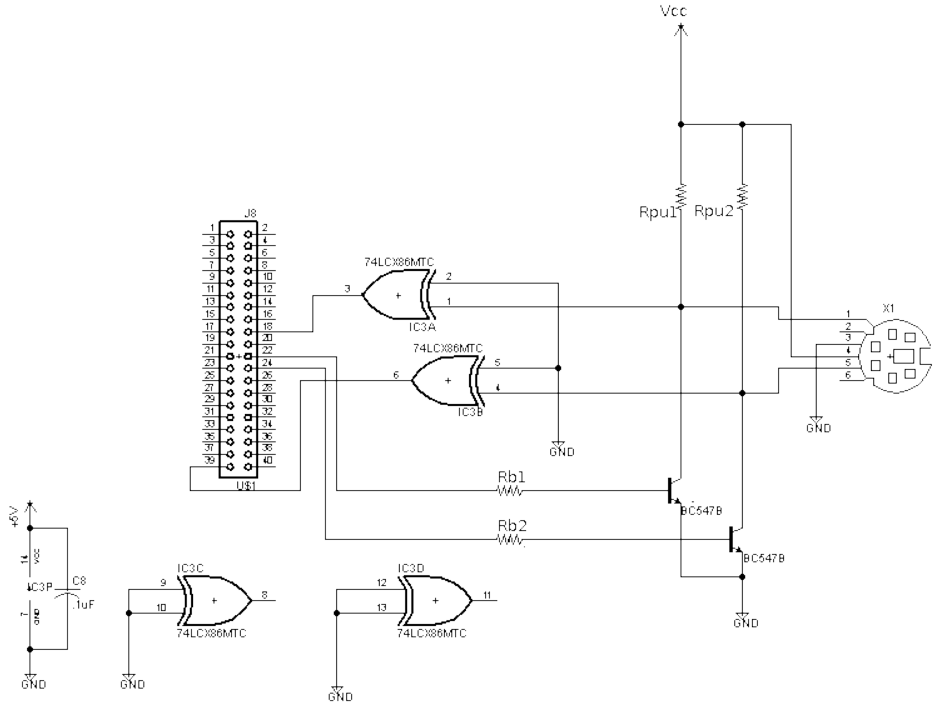


Figure 6. Open Collector circuit for the PS/2 mouse

The PS/2 electrical interface consists of two lines: Data and Clock, pins 1 and 5 from the PS/2 connector respectively. Both lines are open-collector with pull-up resistors to Vcc. An open-collector interface has two possible states: low or high impedance. In the low state, a transistor pulls the line to ground level. In the high state, the interface acts as an open circuit and let the output float. Furthermore, a “pull-up” resistor is connected between the bus and Vcc so the bus is pulled high if none of the devices on the bus are actively pulling it low. The exact value of the pull-up resistor isn’t too important. Typically, the resistances take values between 1 and 10 kΩ.

In order to calculate the right values of its resistors, some rules have been used. The most important things to be aware of are that the pull-up resistors use to be of the order of 10k. The value chosen is 15kohms. Accordingly to this and knowing from the datasheet information that the DC gain of the transistor is approximately 150, the value of the resistors Rb1 and Rb2 have been calculated following the next equations:

According to this, Rb should be:

Experimental tests demonstrated that the Rb values were not good enough, it was too big, and it was decided to reduce its value until getting the right behavior of the circuit. The differences between the calculated value and the final experimentally obtained value used can be caused for many reasons. The first one is that the real DC gain from the transistor, is usually smaller than what is said on the datasheet.

Another important factor is that this resistance is connected to the I/O connector from the demo board, so there can be surges currents from the board itself, unknown from an external observer and difficult to calculate.

The final values of the open-collector components experimentally obtained are:

$R_{pull\text{-}up1} = R_{pull\text{-}up2} = 15\ k\Omega$

$R_{b1} = R_{b2} = 22\ k\Omega$

According to this and following the equations used previously, that resistor values lead us to different base currents.

The collector current is still approximately 327 µA.

The base current is approximately 32 µA.

With those values, the transistor would be working with a DC gain of approximately 10.

### MAX 3224ECAP: RS-232 Transceivers with AutoShutdown Plus

The MAX3224ECAP [27] is an RS-232 Transceiver from MAXIM [28]. It is 3V powered and has automatic shutdown/wakeup features, high data-rate capabilities, and enhanced electrostatic discharge (ESD) protection.

Some electrical features from of the RS-232 transceiver can be obtained from the datasheet. The main features are:

| Parameter | Min | Typ | Max | Unit |
|---|---|---|---|---|
| Supply current |  | 1 | 10 | µA |
| Vcc | 3.3 |  | 5.5 | V |
| Maximum Data Rate | 250 |  |  | Kbps |

Table 4 Electric features of MAX3224E

The AutoShutdown feature allows saving power. It consists on automatically entering a low power shutdown mode if the device finds that the RS-232 cable is disconnected of the transmitters or if there is no activity during more than 30 seconds. It turns on again when senses activity at any transmitter or receiver input.

Other interesting features from this device and accordingly to the datasheet are:

“...The transceivers have a proprietary low-dropout transmitter output stage enabling true RS-232 performance from a +3.0V to +5.5V supply with a dual charge pump. Each charge pump requires a flying capacitor (C1, C2) and a reservoir capacitor (C3, C4) to generate the V+ and V+ supplies. Those capacitors have to be 0.1 µF for operation from a 3.3V supply.

The MAX3224E feature a logic-level output (READY) that tells when the charge pump is regulating and the device is ready to begin transmitting. The READY output is low when the charge pumps are disabled in shutdown mode. The READY asserts high when V- goes below -4V...”

Moreover, recommendations from the datasheet are taken, so for the design, a 0.1 uF Vcc bypass capacitor will be used. This capacitor is used to decouple power-supply. MAX3224E device is 20-pin SSOP packaged.

### 3.2.3 Design of the schematic

Knowing all the parts and the right values of the components needed, the schematic can be designed. The size is not an important factor, but it is intended to design it as small as possible.

With all the information gathered one have to think about how to place each part and have to be careful and try not to overload the tracks between the components. For that I am using a special CAD tool named Eagle Layout Editor [29], which is free software and easy to use.

The schematic design of the IPM system looks like the following:

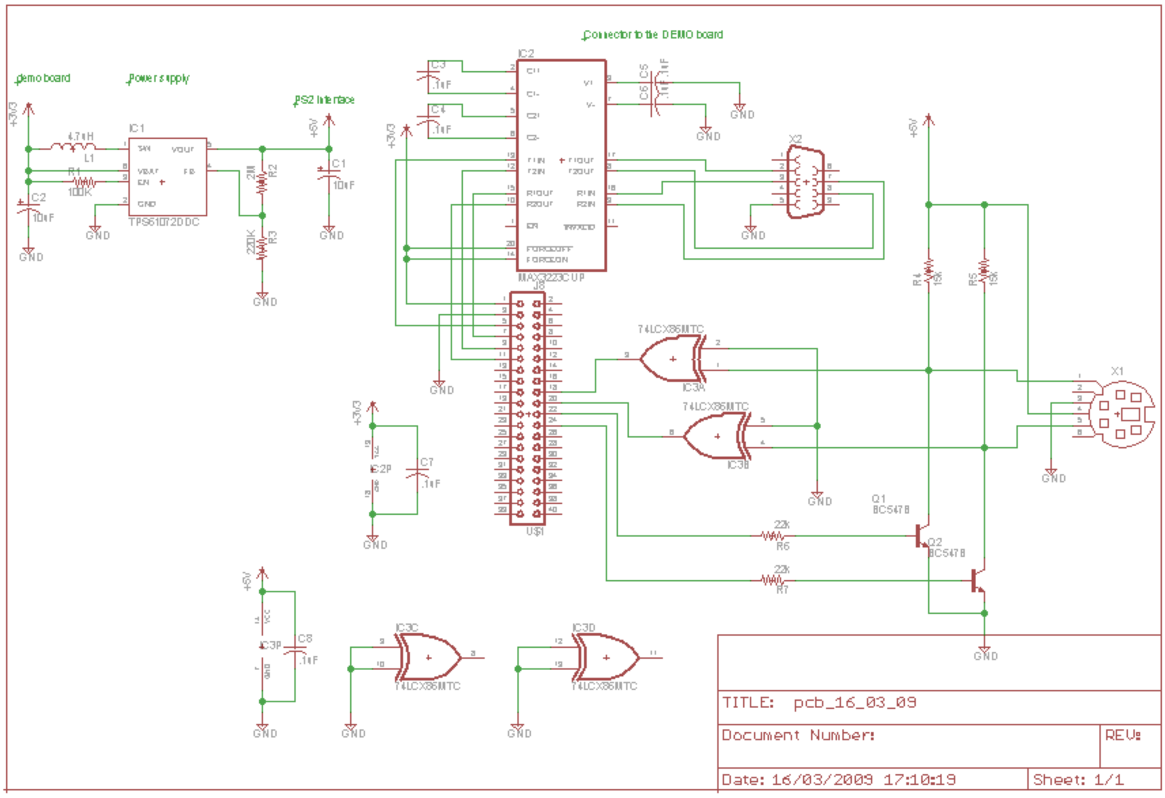


Figure 7. IPM schematic design

### 3.2.4 Design of the PCB

Once the schematic design is finished and without any errors of connection (Eagle has the option to check for errors of connection), the Printed Circuit Board (PCB) needs to be created. For that, Eagle CAD software will be used again. In order to design and place the components on the PCB, some layout considerations need to be followed.

**Layout Considerations for TPS6107x boost converter:**

The layout is an important step in the design, especially when working at high-peak currents and high switching frequencies, like switching power supplies. If the layout is not carefully done, the regulator could show stability and EMI problems, as mentioned in section 3.2.2.1.

Therefore, wide and short traces should be used for the main current path and for the power ground tracks. Moreover, the input capacitor, output capacitor, and the inductor should be placed as close as possible to the Integrated Circuit (IC) ground noise.

The feedback divider should be placed as close as possible to the ground pin of the IC. To lay out the control ground, short traces should be used as well, and they should separated from the power ground traces to avoid ground shift problems, which can occur due to superimposition of power ground current and control ground.

Knowing this, the connections for the PCB can be done. In Eagle, there is a function that takes the schematic information and makes the connections between the components automatically. So the only thing missing to do is to place each component on the right position and route the traces between components, trying to optimize the size of the board, it should be as small as possible, and without crossing or overloading any of the traces, as the circuit would be shorted.

This is not as simple as it seems, because you need to make some intends and replace over and over the components in order to get it.

The final printed circuit board sizes *57.6 x 70.8 mm* and it is shown at figure 8. Note that the board has both top and bottom layers, because this was the way to avoid overloading the footprints and traces between the different components. Red lines are placed on the top layer, and blue lines and devices are placed on the bottom layer. There are also some vias that connect top and bottom layers in order to simplify or short some tracks. Once the board is fabricated, there is a need to connect the top layer with the bottom layer putting a piece of cable inside the hole of the vias, in order to keep the connections between the two layers.

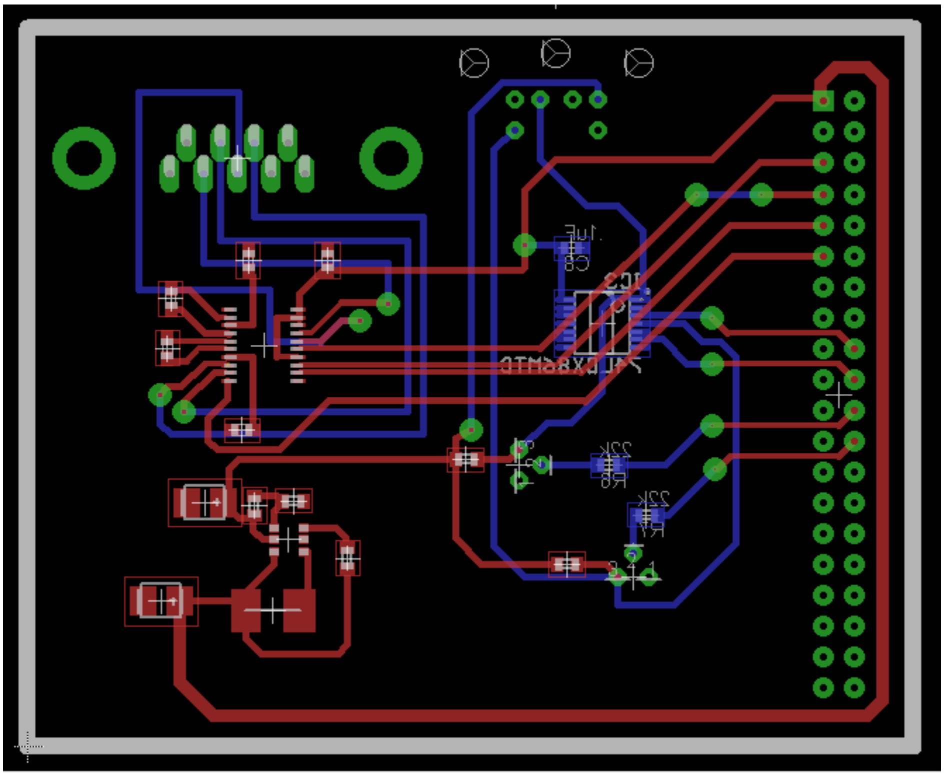


Figure 8. IPM's PCB design

### 3.2.5 Building the board

With the schematic and the PCB designed, the only thing missing is to fabricate the board. To do that, two different programs are used, the CircuitCAM [30] and the Board Master [31]. The machine used to build the board is the LPKF protomat S42 [32].

After creating the Gerbo files, obtained from the same software named before, Eagle, importing the files to the CircuitCAM program and making all the necessary configurations, it will create a new file that contains all the information needed to build the board. This file has an extension (*.LMD).

This file obtained has to be imported to the second program, the MasterBoard. Once there, and with the board printed on-screen, one should check that have the right information and holes size in each layer, and proceed on building the board. It is a slow process, because the drilling tools need to be changed manually for each hole-size, and the two layers, top and bottom, need to be performed separately.

The PCB resulting is the next:

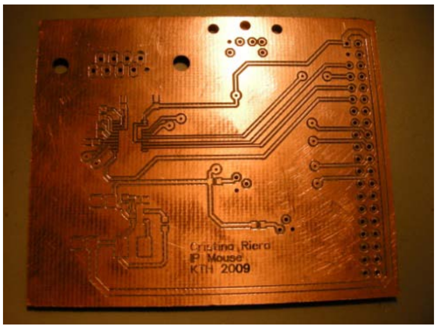


Figure 9. Front side of the PCB

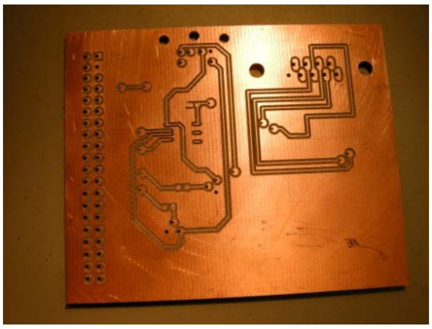


Figure 10. Back side of the PCB

After soldering all the components and vias carefully, the final result of the board is this one:

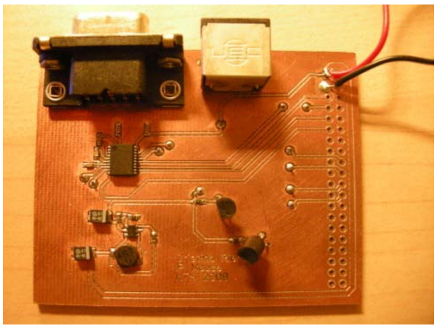


Figure 11. Front side of the final PCB

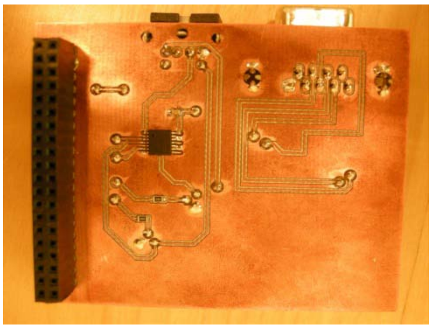
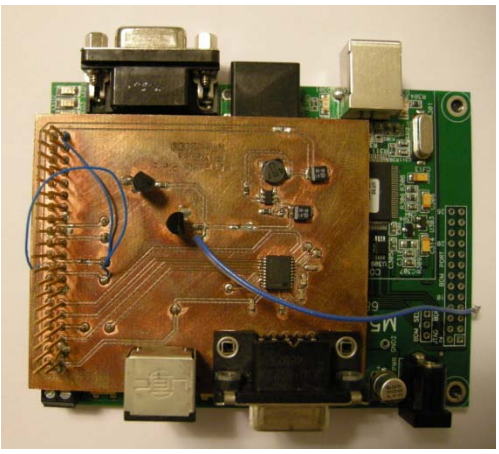


Figure 12. Back side of the final PCB

Attaching the PS/2 interface board with the previously mentioned M52233EVB, the complete system mounted is shown below. On the profile picture of the IPM (figure 14), the PS/2 connector where the external mouse will be connected is shown.



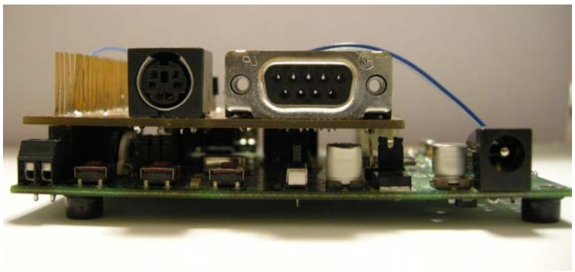


Figure 14. Profile of the complete system

Figure 13. Complete system from the top

# 4 Implementation

The IPM has been designed and built. As any peripheral device, IPM needs to be implemented according to a protocol. Prior to the implementation itself of the device, a good start point is to investigate and search about the actual devices and the protocol they use. A good start point is to search about Human Interface Devices (HID) to find out how computers use them and learn about the protocol they use. The next subsection of this chapter introduces the X-Windowing, which will be useful to learn and understand how windowing systems works and how could that help an IP Mouse device. There is also a subsection introducing the network packets and the TCP/IP stack. Further to those introductory subsections, the IPM implementation itself will be accurately explained.

## *4.1 Human Interface Devices (HID)*

A human interface device (HID) [33], as its name indicates, is a type of peripheral device that enables humans to interact directly with a computer. The most common examples of HID devices are a mouse, a keyboard or a joystick.

HID were intended to innovate in computer input devices by simplifying the process of installing these peripherals. Prior to HID, devices usually conformed to very restrictive protocols that needed upgrading of data and protocol each time the hardware was modified.

By contrast, at present most HID devices have self describing packages that may contain a huge variety of data types and formats, which make the installation much easier. A single HID driver on the PC currently analyzes the data and enables dynamic association of data I/O with application functionality. This has enabled rapid innovation and proliferation of new human interface devices.

There are many types of HID available for modern PCs. The most common interfaces they use to communicate with a personal computer are the Universal Serial Bus (USB) or the Personal System 2 (PS/2). As PS/2 interface has been chosen for this project, it is further detailed on the next subsections.

### 4.1.1 The PS/2 Interface

PS/2 stands for Personal System 2 [2]. The PS/2 system was the second generation line of computers of the IBM Corporate back in the 90's. The PS/2 system was innovative for its age and incorporated a new I/O connector for peripherals such as keyboard and mouse, which later on became a standard connector that was also named as PS/2 connector. PS/2 port is bidirectional and uses an in/out clock to transmit and receive data. Data is sent and received synchronous to this clock.

The PS/2 mouse interface uses a bidirectional serial protocol to transmit data to the host's controller. The computer, in turn, may send a number of commands to the mouse to set the report rate, set the resolution, reset the mouse, enable/disable data reporting, get the device ID, enable/disable the mouse or set mouse mode. The computer also provides the mouse with an overload-protected 5V power supply.

| | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|---|---|---|---|---|---|---|---|---|
| Byte 1 | Y overflow | X overflow | Y sign bit | X sign bit | Always 1 | Mid. Btn | Right Btn | Left Btn |
| Byte 2 | X movement | | | | | | | |
| Byte 3 | Y movement | | | | | | | |

Table 5. Movement Data Packet

Table 5 shows a movement data packet from a standard PS/2 mouse. A standard PS/2 mouse sends a 3 byte packet of movement/button information to the host.

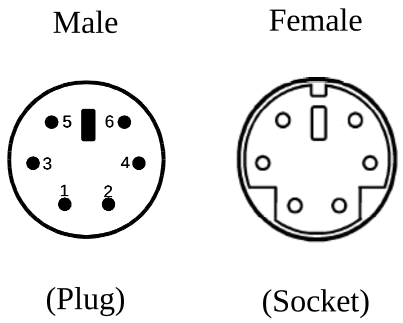
The first byte contains position information, status of the mouse buttons (1 for pressed and 0 for unpressed) and the X and Y overflows, which are set if the counters go past 255. Second and third bytes contain information regarding the X and Y movement respectively.

These movements are calculated based on the counter values in the mouse. These counters are updated when the mouse reads its input, from its sensors, and finds that movement has occurred. Their value is the amount of movement that has occurred since the last movement data packet was sent to the host. The range of values of the movement counters is -255 to +255. If this range is exceeded, the appropriate overflow bit is set.

The resolution is the parameter that determines the amount by which the movement counters are increased / decreased. The default resolution is 4 counts/mm and the host may change that value communicating with the mouse accordingly. The PS/2 mouse also allows scaling the value of the mouse movements. By default the scaling factor is set to 1.

### 4.1.2 PS/2 General information

There are two types of PS/2 connectors: The 5-pin DIN and the 6-pin mini-DIN. Both connectors are electrically similar and the only difference between them is the arrangement of pins. The 6-pin mini-DIN is the one that will be used on the IPM design.



| Pin | Function |
|---|---|
| 1 | Data |
| 2 | Not implemented |
| 3 | Ground |
| 4 | Vcc ( +5V ) |
| 5 | Clock |
| 6 | Not Implemented |

Figure 15. 6-pin Mini-DIN (PS/2) connectors

- Vcc/Ground provides power to the device.
  - Typically: Vcc comprehends values between 4.5 V and 5V.
- The current of the keyboard/mouse must not be higher than 275 mA.
  - That is important to avoid transient surges, caused by “hot plugging” the device (i.e. connect or disconnect the device while the computer’s power is on ).

### 4.1.3 PS/2 Communication Protocol

The PS/2 electrical interface consists of two lines: Data and Clock. Both lines are open-collector with pull-up resistors to Vcc. The open-collector interface has been already presented on section 3.2.

The PS/2 mice implement a bidirectional synchronous serial protocol. It can either transmit data from the Device to the Host and the opposite way, Host to Device. The bus is "idle" when both data and clock lines are high (open collector). This is the only state in which the mouse is allowed to begin transmitting data. The host has ultimate control over the bus and may inhibit communication at any time by pulling the Clock line low.

The device always generates the clock signal. If the host wants to send data, it must first inhibit communication from the device by pulling the clock line low. The host then pulls the data line low and releases the clock. This is the "Request-to-Send" state and signals the device to start generating clock pulses. The device bus states are:

| Data | Clock | State |
|---|---|---|
| High | High | Idle state |
| High | Low | Communication inhibit |
| Low | High | Host Request to Send |

Table 6. PS/2 protocol. Bus states

All data is transmitted one byte at a time. Each frame has a length of 11-12 bits:

- Start bit. This is always 0
- 8 data bits, least significant bit first
- 1 parity bit (odd parity)
- 1 stop bit. This is always 1
- 1 acknowledge bit (host to device communication only)

The parity bit is used for error detection. It is set if there is an even number of 1's in the data bits. Otherwise it is reset. The number of 1's in the data bits plus the parity bit always add up to an odd number (odd parity).

The data sent from the host to the device is read on the rising edge of the clock. Data sent from the device to the host is read on the falling edge. The mouse always generates the clock signal, but the host always has ultimate control over communication.

### 4.1.3.1 Device to Host Communication

Data and Clock lines are both open collector, so the idle state of the bus is high. When the mouse wants to send information, it first checks if the Clock line is at high logic level. If it's not, the host is inhibiting communication and the device must wait until the host releases the Clock line. Prior the device can begin transmitting its data, the Clock line must be at high level for at least 50 microseconds.

The mouse writes a bit on the Data line when the Clock is high, and it is read by the host when Clock is low.

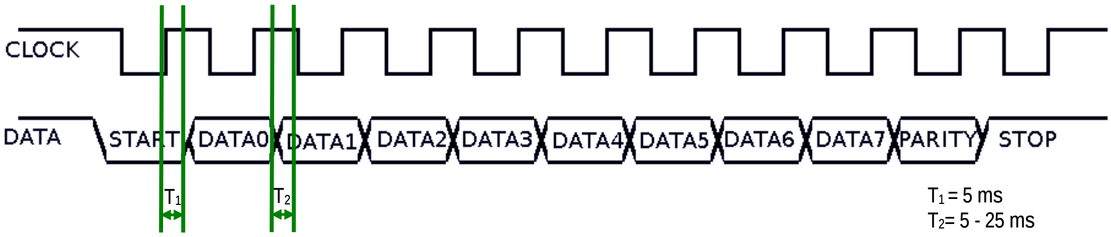


Figure 16. Device to Host data packet and timing diagram

The clock frequency is 10-16.7 kHz. The time from the rising edge of a clock pulse to a Data transition must be at least 5 microseconds. The time from a data transition to the falling edge of a clock pulse must be at least 5 microseconds and no greater than 25 microseconds.

The host may inhibit communication at any time by pulling the Clock line low for at least 100 microseconds. If the communication is inhibited before the 11th clock pulse, the device must abort the current transmission and prepare to retransmit the current data.

If the host pulls the clock low before the first high-to-low clock transition, or after the falling edge of the last clock pulse, the mouse does not need to retransmit any data. However, if new data is created and needs to be retransmitted, it will have to be buffered until the host releases the Clock.

The basic process and timing that the mouse uses while sending a data is:

1. Waits for Clock to be high
2. Delays 50 µs
3. Check if the clock is still high. If it's not, go to step 1
4. Check if data line is high. If it is not, abort communication and read from the host
5. Delays 20 µs (or 40 µs if the time clock is pulled low in sending the start bit)
6. Outputs Start bit (it is always 0)
7. Outputs 8 data bits
8. Outputs Parity bit (odd parity)
9. Outputs Stop Bit (it is always 1)
10. Delays 30 µs (or 50 µs since the time Clock is released in sending the stop bit)

#### 4.1.3.2 Host-to-Device Communication

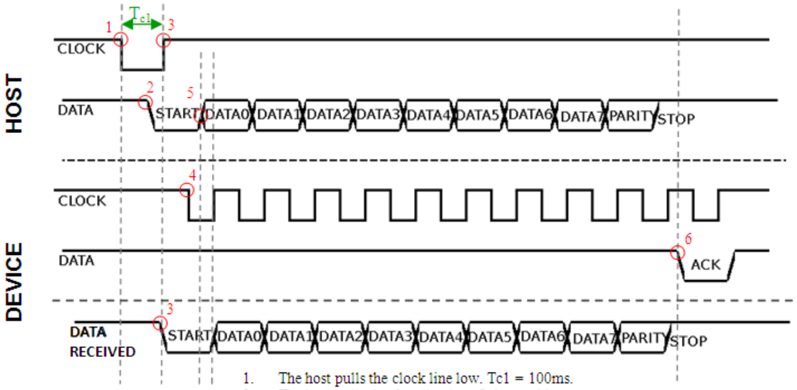
The packet is sent a little differently in host-to-device communication. Since the device always generates the clock signal, if the host wants to send data, it must first inhibit communication and second apply "Request-to-Send" state by pulling Data low, and releasing then the Clock.

The device should frequently check for this state. When the device detects this state, it will begin generating Clock pulses and will send an acknowledge frame with eight data bits and one stop bit. Contrary with Device To Host Communication, in Host to Device communication the host writes the data only when the Clock line is low, and the device reads the data when the clock line is high.

After the clock is received, the device will acknowledge the received byte by bringing the Data line low and generating one last clock pulse. If the host does not release the Data line after the 11th clock pulse, the device will continue generating clock pulses until the Data line is released and then will generate an error.

The host may abort transmission at time before the 11th clock pulse (acknowledge bit) by holding Clock low for at least 100 microseconds. The steps the host must follow to send data to a PS/2 driver are:

1. Bring the Clock line low for at least 100 microseconds
2. Bring the Data line low
3. Release the Clock line
4. Wait for the device to bring the Clock line low
5. Set or reset the Data line to send the first data bit
6. Wait for the device to bring Clock high
7. Wait for the device to bring Clock low
8. Repeat steps 5-7 for the other seven data bits and the parity bit
9. Release the Data line
10. Wait for the device to bring Data low
11. Wait for the device to bring Clock low
12. Wait for the device to release Data and Clock



1. The host pulls the clock line low. Tc1 = 100ms.
2. The host pulls the data line low. It is start bit
3. When the clock line is high, the device reads the data.
4. The device starts generating the clock
5. The host starts sending the 8 bits of information
6. When thehost has sent the stop bit, and the mouse receives it, the device sends an ACK.

Figure 17. Host to Device data packet and timing diagram

### 4.1.3.3 Defining device's operation mode and parameters[1]

As described previously, the host needs to communicate with the device in order to define the parameters the device will use. The device has four standard modes of operation: *Reset*, *Stream*, *Remote* and *Wrap*. They all can be entered by sending the right command to the mouse.

The mouse only enters the **Reset mode** at power-up or after receiving the Reset (0xFF) command. On this mode a self-test is initiated, the mouse sets default configurations values: Sampling rate = 100 samples/s; Resolution = 4 counts/mm, Scaling = 1:1 and Disable Data Reporting. Then responds with an acknowledge (0xFA) command.

The default mode is the **Stream mode**. It is entered after the reset mode is executed. In this mode, the programmed sample rate is the maximum rate of transfer. Data report is transmitted if an event (button pressed or movement detection) occurred.

In the **Remote mode** the data is transmitted only in response to a Read Data command. This mode may be entered by sending the Set Remote Mode (0xF0) to the mouse.

**Wrap mode** is used for testing the connection between the mouse and its host. Wrap mode may be entered by sending the "Set Wrap Mode" (0xEC) command.

There are many commands the host can send to a standard PS/2 mouse in order to set the operating mode and configurations. The IPM is working according to the Stream Mode. The most used and useful for the IP Mouse system are explained below.

- Reset (0xFF): this command is used to enter the Reset Mode. After this command the mouse responds with an "acknowledge" (ACK).

[1] All data information and commands from this section is adapted from the Adams text, that can be found on www.computer-engineering.com

- Set Defaults (0xF6): this command is used to load the default values of the mouse, reset its movement counters and enter to the stream mode. The mouse responds with an ACK.
- Disable Data Reporting (0xF5): is used to disable the data reporting and reset the movement counters. The mouse responds with an ACK.
- Enable Data Reporting (0xF4): is used to enable data reporting and reset its movement counters. The mouse responds with an acknowledge command.

There are many other commands that can be used either to set the sample rate, resolution or scaling of the mouse, to get the device ID or to request the status of the mouse.

The only commands the standard PS/2 mouse will send to the host are “Resend” (0xFE) and “Error” (0xFC). They both work the same as they do as host-to-device commands.

## *4.2 Windowing Systems*

A windowing system [1] enables the computer user to work with multiple applications at the same time. Each program runs in its own window. Most windowing systems have basic support of re-parenting which allows windows to overlap, however the window manager is who controls the way in which windows interact.

Different Windowing systems have been developed until now. Some of them, like X Windowing, are characterized for being network transparent, allowing the user to display graphical applications running on a remote machine. Even though that many windowing systems have been designed over the years, the most known and used at present are Mac OS X, X Window System or Y Window System. Moreover, some operating systems such as Microsoft Windows [34], Mac OS [35] and Palm OS [36], contain windowing systems integrated with the OS.

### 4.2.1 What is X-Windowing?

The MIT [37] team developed X Window as a distributed, network-transparent, device independent, multitasking windowing and graphics system. As a windowing system it permits to display multiple applications on the same screen, and it lets one application use many windows. It supports overlapping and hidden windows, text with soft fonts, and two-dimensional graphics drawing.

X Window uses a client/server model in which, by contrast with the common configuration of client/server systems, the server runs on the local machine (user’s computer), and the client’s applications or requests can run either locally or remotely.

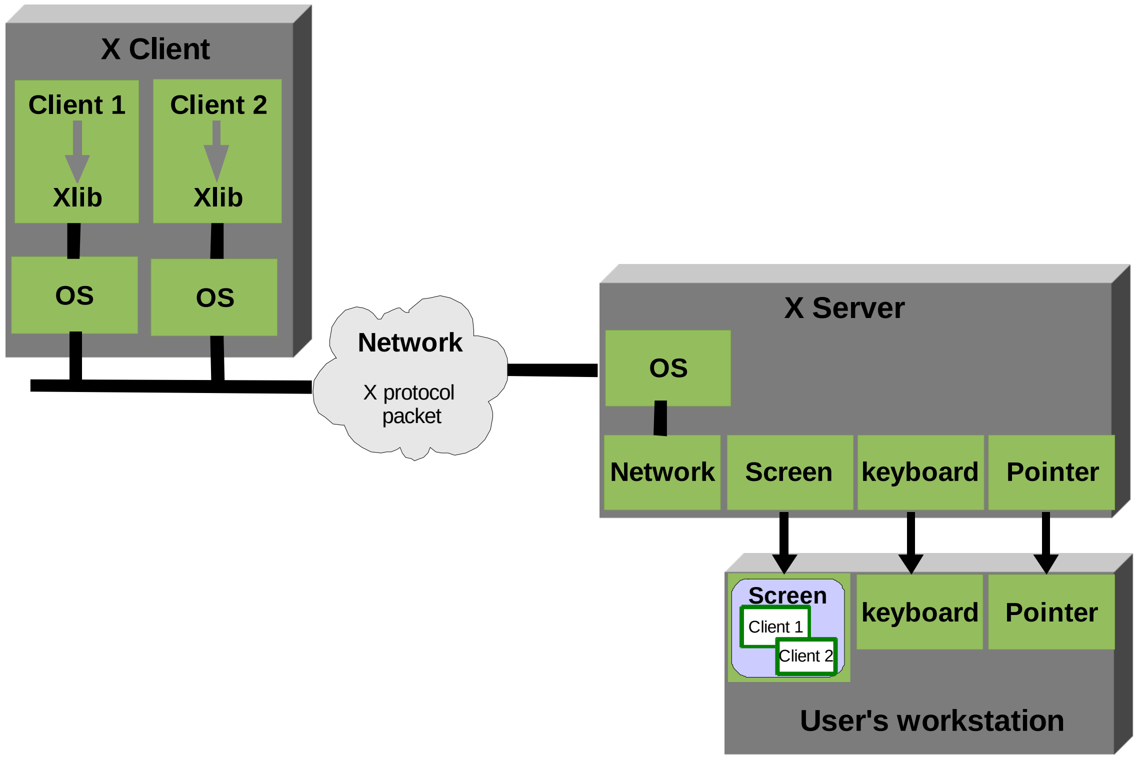


Figure 18. X-Windowing client/server model

The client communicates with the server by sending packets of instructions conforming to the X Protocol. This is a network protocol that can run locally or through a network regardless of whether a client program is local or remote.

Each workstation has its own server, which contains the hardware-dependent drivers for that workstation. The server also manages the requests from the clients: the input devices (keyboard or mouse), and the display of the screen (see figure 18).

X provides the basic framework for building Graphic Unit Interface (GUI) environments: interacting with input devices (mouse or keyboard) and drawing or moving windows on the screen. The application programmer links the client program with X Window using Xlib [38], a library of graphics and windowing functions.

### 4.2.2 The X server

Each workstation has its own X server. The X server interacts with the hardware and performs the action. It captures input events from the input devices (i.e. keyboard or mouse) and passes the information to the Window Manager, which tells the server which client application has requested it and where to replace the window. It also receives requests from the graphical output (window) and sends it back user input (input device).

A good thing from X windowing is its ease of communication. As long as the X clients are written accordingly to the X Protocol, they can run on any system and communicate with the X server. Only the server software has to be hardware-specific.

An X server, from a structural point of view is conformed of:

- A device-independent layer, which receives and translates client request messages.
- An operating system-dependent layer, that interfaces to a particular OS
- A device-dependent layer, where are the device drivers for the specific hardware supported.

### 4.2.3 X-Window System Core Protocol

The communication protocol between server and client runs network-transparently, which means that the client may run on the same machine than the server or on different ones, possibly with different architectures and operating systems.

Communication between server and clients is done by exchanging packets over a network channel. The client is who establish the connection by sending the first packet. The server then sends back a packet stating the acceptance or refusal of the connection. The server can also answer with a request for a further authentication.

If the connection is accepted, the acceptance packet contains data that the client will need to use for the next interaction with the server.

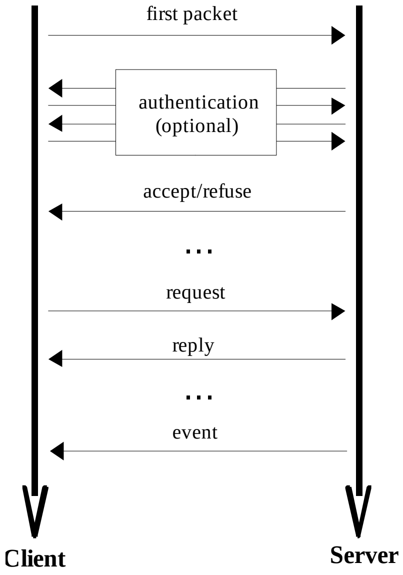


Figure 19. Communication interaction between a client and a server

After connection is established, the client and the server can exchange four types of packets over the channel: *Request*, *Reply*, *Event* and *Error*.

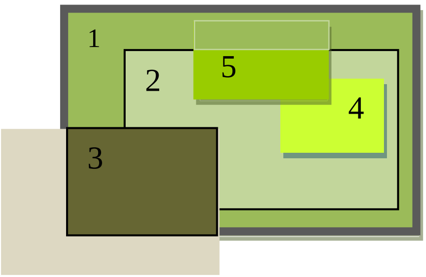
1. *Request*: The client requests information from the server or requests it to perform an action.
2. *Reply*: The server answers to a request. Not all requests generate replies.
3. *Event*: The server sends an event to the client, e.g. keyboard or mouse input, or move, resize or expose a window.
4. *Error*: If a request is invalid, the server sends an error packet. Since requests are queued, error packets may not be sent immediately.

### 4.2.4 Windowing hierarchy

X treats windows as a hierarchy. The root or origin window on which the application windows are displayed is called *parent*. Application windows can only be created as a sub window of a parent window and the name given to them is *children*. In X-Windowing system, graphical elements and input events such as buttons' status, menus, icons or inputs from a peripheral device are all realized using windows.

The root or parent of this windowing hierarchy is automatically created by the server. Moreover, this hierarchy can have top-level windows, which are the direct sub windows of the root window. Each window and sub window may have its own sub windows.

Visibly, the root window is the same size as the screen, and lies behind all other windows. Children always stay in front of their parents. If there's a part of a window outside its parent, this part is not visible. Window manager is who controls the overlap and visibility of windows.



In this figure:

1 is the root window, which covers the whole screen;

2 and 3 are top-level windows (children of 1)

4 and 5 are sub windows of 2 (children of 2)

Figure 20. X-Windows hierarchy

The number of windows that can be created and destroyed is almost limitless. Each window has its own attributes such as foreground, background, border colour, cursor shape, etc.

### 4.2.5 Events

Events are packets sent from the server to the client to notify that something has occurred. Every event is relative to a window, and each packet contains an identifier of that window. Events can be for input (i.e. when a user presses a mouse button) or for output (i.e. to indicate the creation of a new sub-window).

Events can also be used for communication between clients. For example, a client can request the text that is being selected, so this event is sent to the client that is using the window with the selectioned text.

Client requests are sent continuously but the Window Manager is who has ultimate control over the screen usage. The manager tries to satisfy these client requests as fairly as it can. Windows can be moved, resized, closed, or reduced to an icon via the performance of a window manager. In fact, a window manager can do whatever its implementer can dream up.

## *4.3 Network packets*

A network packet is the basic unit of information that is transferred across an Ethernet network. The basic packet consists of a header and a body. The header contains the sending and receiving systems' addresses, and the body contains the data to be transferred. As the packet travels through the TCP/IP protocol stack, the protocols at each layer either add or remove fields from the basic header. When a protocol on the sending system adds data to the packet header, the process is called data encapsulation.

Network packets are important for this work, because the pointing device being created, IPM, will need to send its data through packets via the TCP/IP stack. The data packets need to be created from the beginning, adding a header and a checksum for being able to go through all the internet layers. The data sent by the IPM will be sent through internet and it will be able to be received to any computer.

For the IPM implementation to send the data, UDP protocol has been chosen, as it is connectionless protocol and it is easier to implement the sending interface.

An overview of the TCP/IP stack of the M52233EVB is presented below:

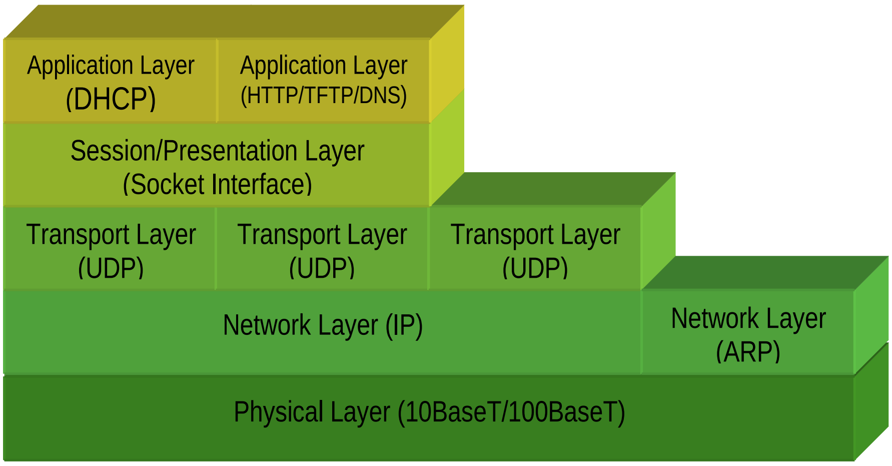


Figure 21. TCP/IP Stack Overview

The Session/Presentation layer is a mini-socket interface similar to the Berkeley Software Distribution (BSD) socket interface. The stack has been optimized for embedded application using zero-copy functionality for minimum RAM usage.

The TCP/IP stack implements the following protocols:

- Internet protocol (IP)
- Internet Control Message protocol (ICMP)
- User Datagram Protocol (UDP)
- Transmission Control Protocol (TCP )
- Ethernet Address Resolution Protocol (ARP)
- Domain Name Server (DNS)
- Dynamic Host Configuration Protocol (DHCP)

For more information about those protocols, the RFC’s official website [39] can be consulted.

## 4.4 Connecting the board

As presented in chapter 3, the M52233EVB is provided with a RS-232 cable, a Code Warrior CD, a USB cable (or debugger cable) and an Ethernet Crossover cable and, of course, the DEMO board, that looks like this:

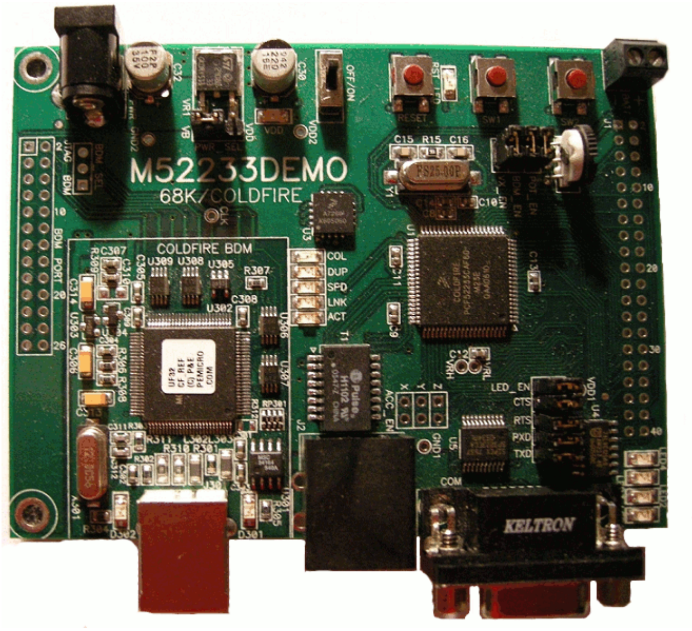


Figure 22. M52233DEMO board

Prior to connecting the board to a computer, specific software needs to be installed on the computer. This software, CodeWarrior, is also provided with the board contents and can be updated through internet.

Once the CodeWarrior is installed, verifying the right configuration of the board and connecting the cables to the computer should be the next steps. Three different cables need to be connected. USB cable will provide power supply to the board. Serial cable will be used for debugging and downloading the code from the computer to the board and Ethernet cable will connect the board to a local Ethernet network. When everything is well connected the two leds situated next to the USB connector turn on (with amber and green colours). More information can be found on the DEMO52233 board quick guide [40].

The board should be now connected and settled. In order to test the right behaviour of that, emulator software should be configured. There are many terminal emulators such as Tera Term, HyperTerminal. Those programs are communications utilities that provide terminal emulation and file transfer protocols, and allow setting options for point-to-point communication via a serial (RS-232) link or a TCP/IP link.

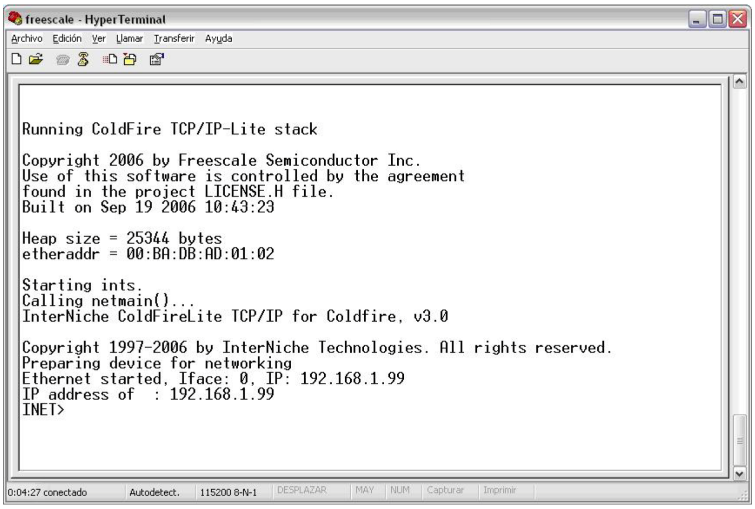
Due to the operative system used is Windows, and it comes with Hyper Terminal included, this is the one that will be used. It can be found by following the next:

Start → Programs → Accessories → Communication → Hyper Terminal

According to the Quick user guide, the terminal has to be set the protocol to 115200, 8 bits, no flow control and 1 stop bit and should as well be configured for the COM port to which the serial cable is attached on the PC. To ensure the right COM port number, Windows have a Devices Administrator place that can be found going to:

Control Panel → System → Hardware → Devices Administrator

Terminal window will display the start-up sequence for the target device. The IP address and Gateway address will also be displayed in a window like this:



```
Running ColdFire TCP/IP-Lite stack

Copyright 2006 by Freescale Semiconductor Inc.
Use of this software is controlled by the agreement
found in the project LICENSE.H file.
Built on Sep 19 2006 10:43:23

Heap size = 25344 bytes
etheraddr = 00:BA:DB:AD:01:02

Starting ints.
Calling netmain()...
InterNiche ColdFireLite TCP/IP for Coldfire, v3.0

Copyright 1997-2006 by InterNiche Technologies. All rights reserved.
Preparing device for networking
Ethernet started, Iface: 0, IP: 192.168.1.99
IP address of  : 192.168.1.99
INET>
```

Figure 23. Hyper Terminal log window

From now on the board is connected and ready to start using and testing the right behaviour of the same. To do that, ColdFire have available some demonstrations and applications that can be used easily by navigating to the IP address 192.168.1.99, that is the demo board IP, and clicking on the links of the applications that appear on the welcome screen display. It is useful to learn and understand the right functioning of the board.

## 4.5 Programming the board

At this point, the hardware part is completely finished but it has to be tested in order to check that it is working alright. The next sections explain the tests performed and used to learn about the functioning of the Freescale board and to get ability to manage the CodeWarrior tools and the board features. By doing these tests, the right functioning and behaviour of the board are checked.

### 4.5.1 Testing the board

The first test consists of implementing two simple programs. First of them is a simple “Hello World”. In order to implement that, a new stationary project needs to be created. As expected, using and executing the code shown below, a new window with a “Hello World in C” written appears, which means that the board is working properly.

```
/** Hello World **/
#include <stdio.h>
int main()
{
      printf("Hello World in C\n\r");
      fflush(stdout);
      while(1);   //idle
      return 0;
}
```

The second implementation done is a looping program. The aim of it is to loop two pins of the I/O port connector of the demo board turning them on and off alternatively. To do that, the ColdFire microprocessor needs to be configured carefully, because many of the pins associated with the external interface may be used for several different functions. When not used for their primary function, many of the pins may be used as general-purpose digital I/O pins (GPIO pins), which means that can be used either as an input or as an output and with digital signal.

Register information from M52233RM [41] provides the information needed to configure the correspondent ports as wished. First off, the I/O ports need to be configured as a GPIO function. The function/variable used to implement that is called PANPAR (IPSBAR:0x10_000A) and it assigns the register bits to the pins. If the bit is 1, the pin assumes its primary function. If it is 0, the pin assumes its GPIO function.

Using the command: MCF_GPIO_PANPAR = 0x00, the program assumes that all the pins are settled as a GPIO function.

Once the GPIO function is settled the data direction of the pins need to be established. The pins need to be set either as input or output.

To do that, the ColdFire has the DDRn register. The DDRn registers control the direction of the port n pin drivers when the pins are configured for digital I/O. The DDRn registers are read/write. At reset, all bits in the DDRn are cleared to 0’s.

If DDRn is 1, the pin is configured as an output. By contrast, if it is 0 it is configured as an input. After using this command, the pins are configured as its GPIO function and as input or output.

For this looping test, pins 22 and 24 (AN6 and AN7 respectively) are configured as outputs and all the others as inputs. This can be done using this command:

```
MCF_GPIO_DDRAN = 0xC0          //  1100 0000
```

The final code for the looping test program is the following:

```
/************************************************
  ************* LOOPÌNG TEST ******************
*************************************************/

#include "mcf5223_gpio.h"
#include "common.h"

int main()
{
      /*PnPARx pin assignment register bits:
                  1 Pin assumes its primary function
                  0 Pin assumes its GPIO function
     */

      MCF_GPIO_PANPAR = 0x00;       // Configuration of PORTAN as GPIO.

     // It could also be bit by bit, but we don't want any Analog I/O.

     /*DDRnx sets data direction for port nx pin when the port is
configured as a digital output
                  1 DDRnx is configured as an OUTPUT
                  0 DDRnx is configured as an INPUT
     */

     // Configuration of pins AN7 and AN6 as an OUTPUT
      MCF_GPIO_DDRAN = 0xC0;


      while(1)
      {

            if(MCF_GPIO_PORTAN == 0x00)
            {

                  MCF_GPIO_PORTAN = 0x40; //AN6 = 1; (logic 1)
            }
            else
            {
                  MCF_GPIO_PORTAN = 0x80; //AN7 = 1; (logic 1)
            }
      }
}
```

The result is a square signal of logic values ‘1’ and ‘0’ in both pins 22 and 24 of the J1 connector, which is the 40-pin header connection mentioned on previous chapter. The signal obtained in this test is the following:

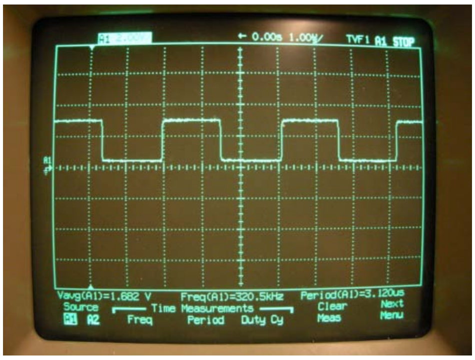


Figure 25. Result of the looping test (pin 22)

Looking at the figure, an observer can see that the output signal is a square signal of approximately 3.2 Vp-p and 320.5 kHz frequency. This frequency might depend on how many instructions are written between the changes of the logic levels, so the output signal may not be constant.

With those two little tests, one can assume that the board is working alright.

### 4.5.2 Interruptions Test Program

For the final program, the pointer device will generate the clock signal. As it has been explained on the PS/2 protocol section (section 4.1) and in order to follow the protocol, some strict conditions need to be considered. For example, in order to synchronize the data to the clock line, which is required to send and read the data from the device, the program may need to know when the clock signal is low, rising, high or falling.

The best way to solve that is by using interruptions. The problem is that the M52233DEMO board does not support interruptions on its I/O data port. The solution is to use an external interruption request (IRQ) from the Edge Port (EPORT) and reconfigure the pins of the connector.

Interrupt Requests (IRQs) are a mechanism that allows the program to jump the processing queue in order to execute the predetermined tasks even if is not their turn.

For the implementation of IP Mouse, the external interruption IRQ7 (pin 39 of the J1 connector of the DEMO board) will be used. This interruption can be configured to stop the running program every certain established time or to sign when the signal is rising or falling. The program will use IRQ7 to sign every time the edge changes its value.

The right commands to be followed in order to configure the interruptions for both rising and falling edges are:

```
#if 1
    /* Enable IRQ signals on the port */
    MCF_GPIO_PNQPAR = 0
        | MCF_GPIO_PNQPAR_IRQ7_IRQ7;

    /* Set EPORT to look for both rising and falling edges */
    MCF_EPORT_EPPAR0 = 0
        | MCF_EPORT_EPPAR_EPPA7_BOTH;

    /* Clear any currently triggered events on the EPORT  */
    MCF_EPORT_EPIER0 = 0
        | MCF_EPORT_EPIER_EPIE7;

     MCF_EPORT_EPFR0 |= 0x80; /** Reset flag


    /* Enable interrupts in the interrupt controller */

    MCF_INTC0_IMRL &=  ~( MCF_INTC_IMRL_MASK7);

#else
```

Typing this code to the *main.c* file and modifying the code provided inside *int_handlers* file to support IRQ7 and recognize the variable *flag_up*, the program looks like it’s showed to the next page:

```
/*
 * File:          main.c
 * Purpose:       sample program
 */

#include <stdio.h>
#include "common.h"

#include "mcf52233_gpio.h"

void configurePins(void);
void configureInterrupts(void);

unsigned char pepi;

int main()
{
      mcf52233_init();
      printf("Write sth!!!! \n");
      pepi = 0;
      while(1){}
      return 0;
}

void configurePins()
{

      printf("Configuring pins ... \n");
      fflush(stdout);

      /* Enable IRQ signals on the port */
    MCF_GPIO_PNQPAR = 0
        | MCF_GPIO_PNQPAR_IRQ7_IRQ7;

      return;
}
void configureInterrupts()
{
      printf("Configuring interrupts ... \n");
      fflush(stdout);

    /* Set EPORT to look for rising and falling edges */
      MCF_EPORT_EPPAR0 = 0
        | MCF_EPORT_EPPAR_EPPA7_BOTH;

    /* Clear any currently triggered events on the EPORT  */
    MCF_EPORT_EPIER0 = 0
        | MCF_EPORT_EPIER_EPIE7;

    MCF_EPORT_EPFR0 |= 0x80;  /** Reset flag

    /* Enable interrupts in the interrupt controller */
      MCF_INTC0_IMRL &=  ~(MCF_INTC_IMRL_MASK7);

      return;
}
```

On the demo board, IRQs pins are connected with the Switches 1 and 2 of the same board. Particularly, IRQ7 is related with SW2. For this test, using the SW2 simulates a manual clock, because its pin is set to ‘1’ or to ‘0’ depending on if the switch is pressed or unpressed.

To try the code, a break point has to be inserted. The correct way to see if the interruption routine is executing and check if the interruptions are working fine is to let the program run and press the button (keeping it pressed). The interruption routine should be then executed, what means that the pin level has changed. Running the program again, and unpressing the button, the interruption routine should be executed as well. As expected, the program stops at both cases, which means that the interruption routine is configured alright for both falling and rising edges.

### 4.5.3 Final program

The aim of the final program is to simulate an implement a mouse according the PS/2 protocol. To do so, the program utilizes the features mentioned in previous sections such as the configuration of the GPIO port and pins and interruption routine. In addition, it needs to have some new functions, to implement the features of being able to read/write data from/to the mouse.

To do that, the PS/2 protocol has to be read carefully again (see PS/2 subsection) in order not to forget anything. The PS/2 mouse protocol is very useful, but to see it clearer a mouse state diagram has been drawn, including all the states that need to pass through to get a good communication in both cases: Host To Device and Device.

After the mouse state diagram, the code use to implement all the states is presented and further detailed.

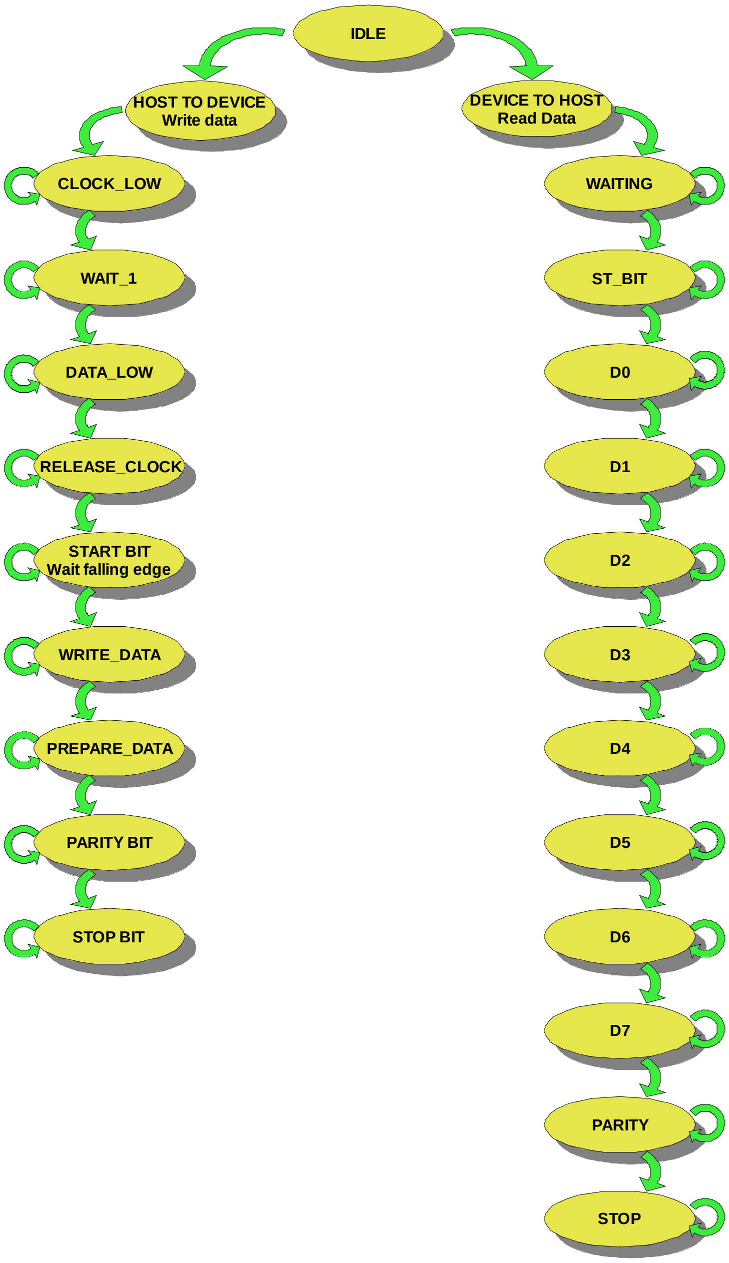


Figure 24. Mouse State Diagram

Having the mouse protocol and the state diagram clear, the final program can be started. Next there's a brief explanation of what is done in each part of the final code and the function of it.

As any usual program, the first thing to do is to include the files and libraries that will be needed altogether with the definitions of the variables that will be used and an introduction of the functions that need to be created as well as declaration and initialization of the main variables.

```
  #include <stdio.h>
  #include "common.h"
  #include "main.h"
  #include "mcf52233_gpio.h"

 #define READ_CLOCK() ((MCF_EPORT_EPPDR0 & MCF_EPORT_EPPDR_EPPD7)>>7)
 #define READ_DATA() ((MCF_GPIO_SETAN & MCF_GPIO_SETAN_SETAN4) >> 4)

  #define RESET_COMMAND 0xff
  #define DATA_REPORTING_COMMAND 0xf4
  #define SET_STREAM_MODE    0xea

  void configureInterrupts(void);
  void sendData(unsigned char Mdata);
  unsigned char parity(unsigned char par);

enum _d2h
{
      INIT,
      WAITING,
      ST_BIT,
      D0,
      D1,
      D2,
      D3,
      D4,
      D5,
      D6,
      D7,
      PARITY,
      STOP;
};

typedef enum _d2h d2hState;

d2hState state;

unsigned char flag_up = 0;
unsigned char pinClock;
unsigned char pinData;
unsigned char data = 0;
```

By these lines:

```
#define READ_CLOCK()((MCF_EPORT_EPPDR0 & MCF_EPORT_EPPDR_EPPD7)>>7)
#define READ_DATA()((MCF_GPIO_SETAN & MCF_GPIO_SETAN_SETAN4) >> 4),
```

the program is reading the correspondent pins where the data from the Mouse is received.

From now on, the program will assume that if it has READ_CLOCK it will read the clock line, and when facing with READ_DATA it will read from the pin in which the data from the mouse is received. Right after that, there are the commands that compulsory need to be sent to the mouse. Its value is defined on the PS/2 mouse protocol.

```
#define RESET_COMMAND 0xff
#define DATA_REPORTING_COMMAND 0xf4
#define SET_STREAM_MODE    0xea
```

The major part of the code is implemented inside the main function. In order to do not mess the program too much, the full system is divided into two principal parts easily distinguishable one from another: Device to Host and Host to device. In addition, some functions are created and used in parallel to the main ones.

The main function is opened with the declarations and initialization of some variables and the configuration of the I/O port into GPIO function, which has already been further detailed on section 4.5.1 Test programs.

```
int main()
{
      unsigned char tick = 0;
      unsigned char Data = 0;
      unsigned char parity = 0;
      unsigned char countB = 0;
      unsigned char Byte1=0, Byte2=0, Byte3=0;

      state = INIT;
      mcf52233_init();

      MCF_GPIO_PANPAR = 0x00;   // Configuration of PORTAN as GPIO.
      MCF_GPIO_DDRAN = 0xC0;    // Configuration of pins AN7 and AN6
                               // as an OUTPUT
      MCF_GPIO_PORTAN = 0x00;   // Initialize M_clock and M_data both
                               // high
      ...
```

Until here, some variables have been defined and the PORTAN, the I/O data port used, have been configured as a GPIO. Direction of the pins has also been configured. Pins 6 and 7 are set as outputs, because the program might need to write on them. Pins from 1 to 5 are configured as inputs, because the program will read from them.

Physically, at this point, both data and clock are at high level, so the device should be now on IDLE state. Following with the code, the next part to program is the Host to Device, where the host needs to send some data to the device in order to establish communication and set previous configurations. This is important because the mouse is previously configured as *disabled data reporting*, which means, the device cannot send any data unless it is told to do so. All this is implemented inside the *sendData* function. The *sendData* function is based on what the protocol says and follows the diagram states shown at Figure 24.

To begin transmitting data, and after initializing the variables *i* and *Mdata2*, the clock has to inhibit communication by pulling the clock low. The delay of 100 ms inserted is because of the protocol. Afterwards, data must be pulled low while holding clock low, and immediately after, the clock line must be released.

The device is now generating the clock signal itself. Its frequency is 12.6 kHz approximately. The variable *flag_up* reads the clock line. Its value is 1 if the clock is at high level and 0 if the clock is low. Since the data is already at low level, the program will assume that it is the start_bit, so the program needs to wait one clock cycle doing nothing, just waiting.

Variable *i* is an auxiliary variable, which will be used to count the number of bits sent. While it is smaller than 8, because the host need to send 8 data bits, and while the clock line is high (*flag_up* = 1), the program just waits, because while the clock is high the host is sending the data, and the device reads this data when the clock is low. At the next falling edge of the clock, the program sends the first bit of data.

*Mdata* is the byte sent by the mouse. By doing `Mdata & 0x01`, the program is taking only the first bit of that byte. If *Mdata2* (the data bit) is different from 0, the system pulls the data high. By contrast, if it is not, the program pulls the data line low.

The next thing the program does is to shift the mask one position in order to get the next bit at the next round, and increases the counter one unity. It stays inside that loop until i = 8 and then waits for the clock line to go high.

After that, if the parity bit is 1, the program pulls data high and if it is not, pulls data low. The program then just waits another clock cycle until it goes high and then pulls data high, which will be used as a stop_bit.

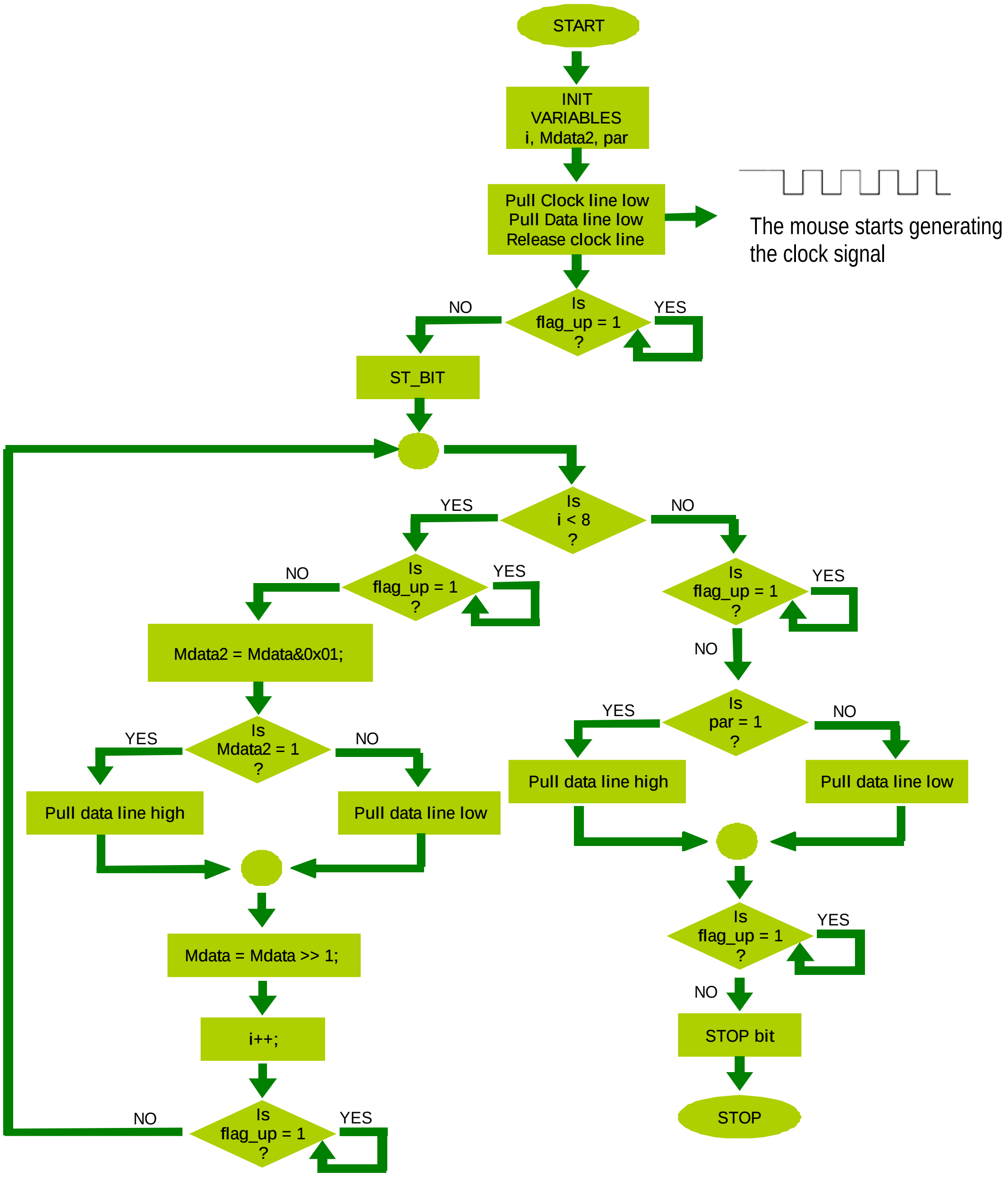


Figure 25. SendData Flow Chart

Figure 25 shows the *sendData* function, and the implementation code is written below:

```
void sendData(unsigned char Mdata)
{
      int i=0, Mdata2 = 0;

      unsigned char par = parity(Mdata);

      MCF_GPIO_PORTAN = MCF_GPIO_PORTAN
                         | MCF_GPIO_SETAN_SETAN7;     //Pull clock low

      cpu_pause(100);         //delay

      MCF_GPIO_PORTAN = MCF_GPIO_PORTAN
                      | MCF_GPIO_SETAN_SETAN6;  //Pull data low while
holding clock low

      MCF_GPIO_PORTAN = MCF_GPIO_PORTAN & 0x7F;       //Release clock

      while(flag_up) {}
      while(!flag_up) {}            // start bit

      while(i<8)
      {
            while(flag_up) {}
            if(!flag_up)
            {
            Mdata2 = Mdata & 0x01;

            if (Mdata2)MCF_GPIO_PORTAN = MCF_GPIO_PORTAN & 0xBF;
            else MCF_GPIO_PORTAN = MCF_GPIO_PORTAN | 0x40;

            Mdata = Mdata>>1;

            i++;
            }
            while(!flag_up) {}
      }
      while (flag_up) {}

            if(par)     MCF_GPIO_PORTAN = MCF_GPIO_PORTAN & 0xBF;
            else MCF_GPIO_PORTAN = MCF_GPIO_PORTAN | 0x40;

      while (flag_up) {}

            MCF_GPIO_PORTAN = MCF_GPIO_PORTAN & 0xBF;
}
```

By this line `unsigned char par = parity(Mdata);` the *sendData* is using the parity function to calculate the parity bit of the bytes sent (Mdata).

The parity bit is used for error detection. It is calculated according to the odd parity. What this function does is to take bit by bit and add them up. If result value is 1, the parity will be 0. If it is 0 the parity bit will be 1.

```
unsigned char parity(unsigned char Mdata)
{
      unsigned char par = 0;
      int p, aux=0;
      for (p=0;p<8;p++)
      {
            aux = Mdata & 0x01;
            par += aux;
            par &= 0x01;
            Mdata>>1;
      }
      par = ~par;
      return par;
    }
```

*SendData* function is used on the main function to send the commands, previously defined, from the computer to the device.

```
// ------------------ HOST TO DEVICE--------------------

      sendData(RESET_COMMAND);      // To set the default values of
the mouse

      sendData(SET_STREAM_MODE);

      sendData(DATA_REPORTING_COMMAND);   //Enables Data Reporting
from the mouse

      state = WAITING;

      flag_up = READ_CLOCK();       //initialized at the current state
of the clock
```

Once the mouse is settled and ready to receive and send data, the Device to Host part is programmed. To implement that part, the protocol and the state diagram are being followed once again. Next graphic is useful to understand what happening and what is the program is doing.

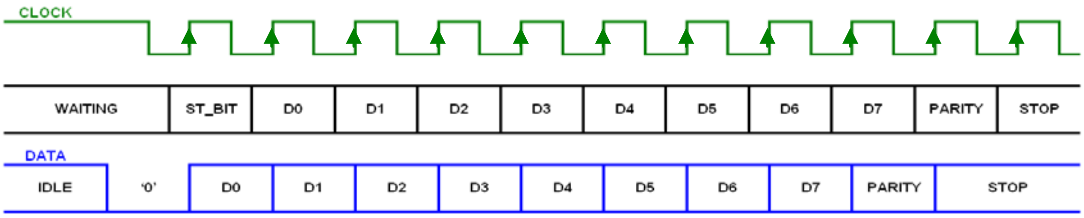


Figure 26. Sending and Reading data from a mouse

On the figure 26 there are the rising edges of the clock marked. The mouse always starts at the "Idle" state, with both clock and data lines high, and waits for something to happen.

The mouse writes a bit on the Data line when Clock is high, and it is read by the host when Clock is low. If the device wants to send data, it must inhibit communication first by pulling the clock line down. Data is sent and received synchronous to the clock.

A flow chart is drawn hereafter to understand the general concept of what the reading of the data is achieved. This shows the main part of reading the data from the mouse, implemented with a switch-case, where each case is one state of the mouse.

Each state is better detailed on the next paragraphs.

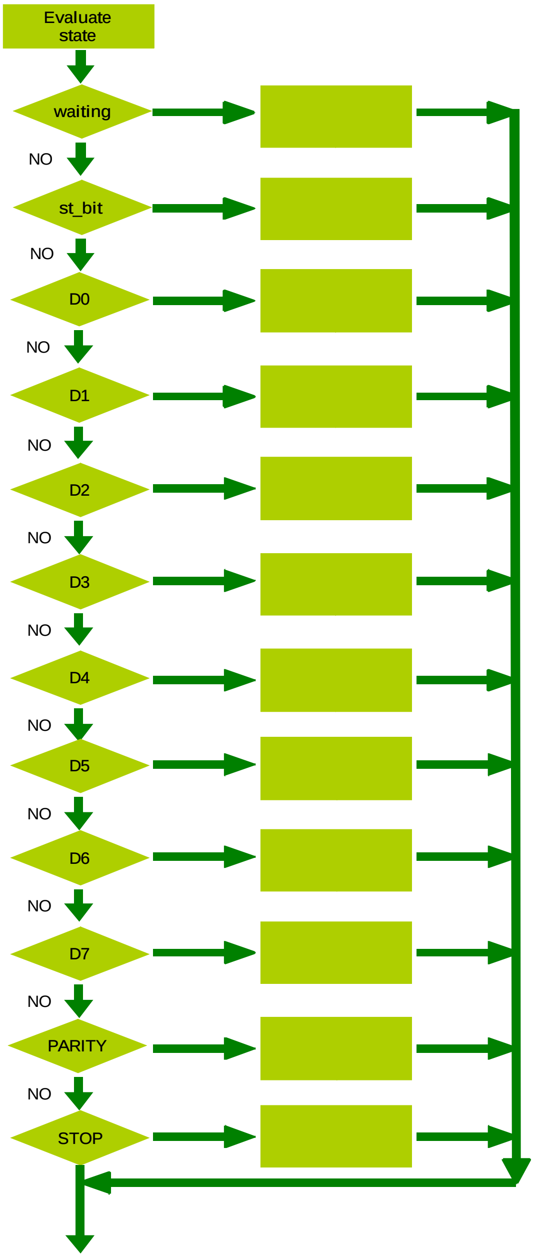


Figure 27. Flow chart of readData

In the waiting state: the program first reads the clock and data line. They both must be at high level. Then do nothing until the clock line falls to a low state. While the clock line is low, the mouse writes the data to be sent. If the data line is low, which means that the START_BIT has been sent, the program initialize variables *parity*, *Data* and *tick* to 0 and passes to the next state: START_BIT. *tick* is an auxiliary variable that changes its value every time the clock changes its edge.

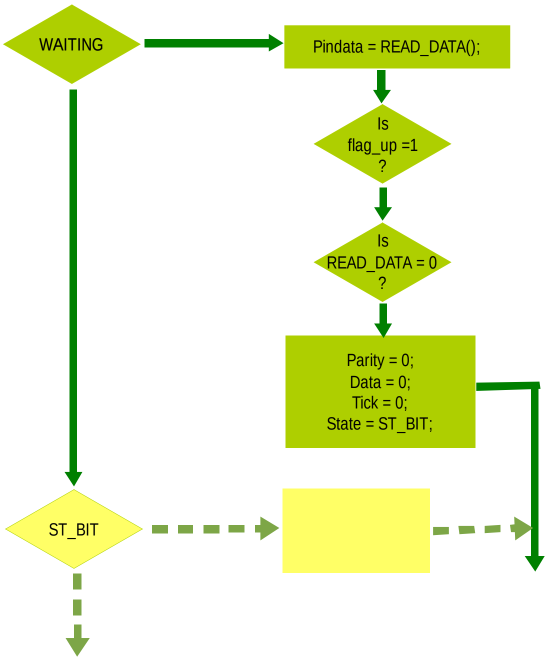
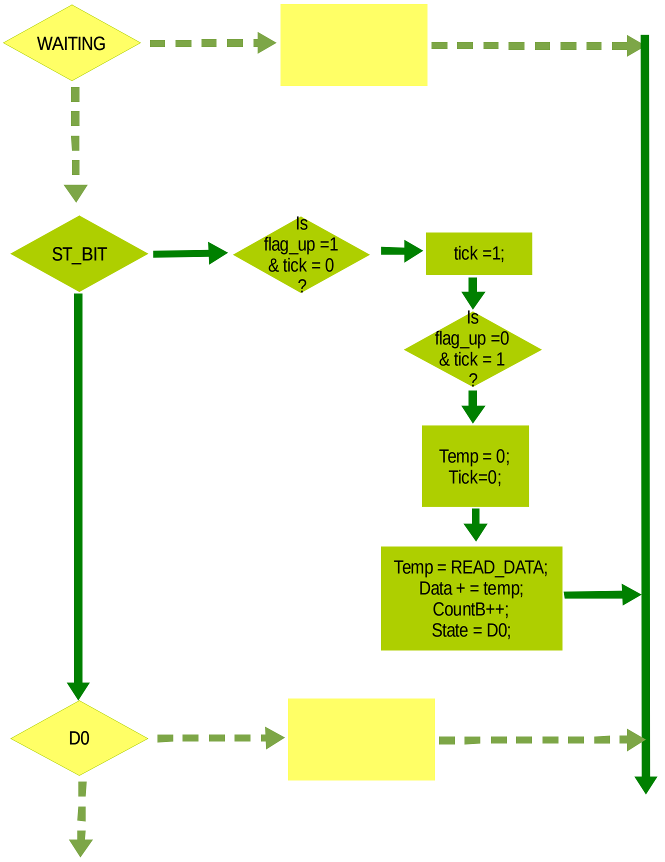


Figure 28. Flow chart for the "waiting state"

```
// ------------------ DEVICE TO HOST ---------------------
while(1)
      {
            switch(state)
            {
                  case WAITING:

                        pinData = READ_DATA();
                        if(!flag_up)
                              if(!READ_DATA())       //START bit
                        {
                              parity = 0;
                              Data = 0;
                              tick = 0;
                              state = ST_BIT;
                        }
                        break;
```

During the START_BIT state, the program reads the clock and the *tick*. When the clock rises (*flag_up* = 1) and the *tick* variable is 0, the program changes the *tick* to 1. Afterwards, initializes the *temp* variable to 0, and changes the *tick* to 0 again. Then the host reads the first bit of data, D0 in this case, and save this bit inside the *Data* variable. Now the value of *Data* = START_BIT + D0. Thereafter the program increases the counter 1 unity and goes to the next state, D0.


WAITING
ST_BIT
Is flag_up =1 & tick = 0 ?
tick =1;
Is flag_up =0 & tick = 1 ?
Temp = 0; Tick=0;
Temp = READ_DATA; Data + = temp; CountB++; State = D0;
D0

```
case ST_BIT:

      if(flag_up && !tick)tick=1;
      if(!flag_up && tick)
      {
            unsigned char temp = 0;
            tick = 0;
            temp = READ_DATA();           //read D0
            Data += temp;
            countB++;
            state = D0;
      }
      break;
```

During the D0 state, if the *flag_up* =1 and the *tick* = 0, *tick* changes to 1.

At the falling edge of the clock, initialize the variables *temp* and *tick* and read the next bit of data, D1. The program shifts it one position in order to put the final data on the right order. The actual *Data* value is START_BIT + D0 + D1.

Once again, the program increases the counter one unity and goes forward to the next state, D1.

From D1 to D6 states, the program is doing almost the same. Change the *tick* variable, wait for the falling edge of the clock and initialize once again the *temp* and *tick* variables.

The host is then ready to read one more bit of data, and place it to its right position. Afterwards, updates the *Data* variable, increases the counter and go to the next state.

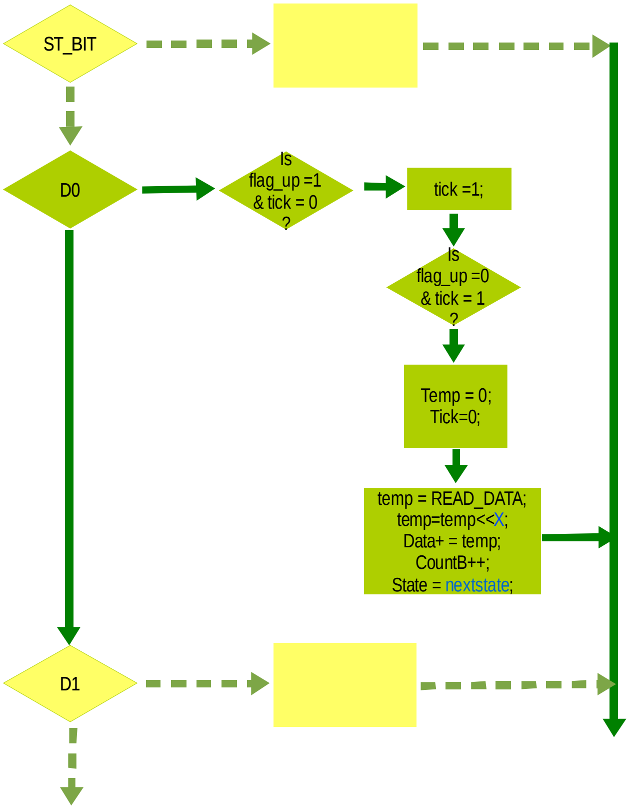


Figure 29. Flow Chart of D0 state

The cases D0 – D6 do exactly the same but only change the X value and the nextstate. In that case, as D0 is being implemented, X should be 1 and nextstate should be D1. The X value increases 1 unity for each state, as the bit taken is 1 position moved. So at D6 state X have to be 7 and nextstate will be D7.

That can be clearer looking at the code:

```
case D0:

       if(flag_up && !tick)tick=1;
       if(!flag_up && tick)
       {
             unsigned char temp = 0;
             tick = 0;
             temp = READ_DATA();     //read D1
             temp = temp << 1;
             Data += temp;
             countB++;
             state = D1;
       }
       break;

 case D1:

       if(flag_up && !tick)tick=1;
       if(!flag_up && tick)
       {
             unsigned char temp = 0;
             tick = 0;
             temp = READ_DATA();     //read D2
             temp = temp << 2;
             Data += temp;
             countB++;
             state = D2;
       }
       break;

 case D2:

       if(flag_up && !tick)tick=1;
       if(!flag_up && tick)
       {
             unsigned char temp = 0;
             tick = 0;
             temp = READ_DATA();     //read D3
             temp = temp << 3;
             Data += temp;
             countB++;
             state = D3;
       }

       break;
```

```
                  case D3:
                        if(flag_up && !tick)tick=1;
                        if(!flag_up && tick)
                        {
                              unsigned char temp = 0;
                              tick = 0;
                              temp = READ_DATA();     //read D4
                              temp = temp << 4;
                              Data += temp;
                              countB++;
                              state = D4;
                        }

                        break;

                  case D4:
                        if(flag_up && !tick)tick=1;
                        if(!flag_up && tick)
                        {
                              unsigned char temp = 0;
                              tick = 0;
                              temp = READ_DATA();     //read D5
                              temp = temp << 5;
                              Data += temp;
                              countB++;
                              state = D5;
                        }

                        break;

                  case D5:
                        if(flag_up && !tick)tick=1;
                        if(!flag_up && tick)
                        {
                              unsigned char temp = 0;
                              tick = 0;
                              temp = READ_DATA();     //read D6
                              temp = temp << 6;
                              Data += temp;
                              countB++;
                              state = D6;
                        }

                        break;

                  case D6:

                        if(flag_up && !tick)tick=1;
                        if(!flag_up && tick)
                        {
                              unsigned char temp = 0;
                              tick = 0;
                              temp = READ_DATA();     //read D7
                              temp = temp << 7;
                              Data += temp;
                              countB++;
                              state = D7;
                        }

                        break;
```

```
      case D7:

            if(flag_up && !tick)tick=1;
            if(!flag_up && tick)
            {
                  unsigned char temp = 0;
                  tick = 0;
                  temp = READ_DATA();     //read parity
                  parity += temp;

                  state = PARITY;
            }

            break;
```

As in all the other cases, the first thing the program does in case D7 is to change the *tick* value to 1 and wait for the falling edge of the clock. After that, it initializes the *temp* and *tick* variables and read the bit data previously sent. This bit is the parity bit. So the program updates this bit at the parity variable and changes the case to the PARITY case.

```
case PARITY:

            if(flag_up && !tick)tick=1;
            if(!flag_up && tick)
            {
                  unsigned char temp = 0;
                  tick = 0;

                  temp = READ_DATA();     //read STOP bit

            if (countB==8)
            {
                  Byte1 = Data;

            state = WAITING;
            }
            if (countB==16)
            {
                  Byte2 = Data;

            state = WAITING;
            }
            if (countB==24)
            {
                  Byte3 = Data;

                  countB = 0;
                  state = STOP;
            }

            }
            break;
```

The PARITY case starts the same as the previous ones, setting the *tick* and waiting for the falling edge and initializing *temp* and *tick* variables again. Then, one more data bit is read, and since it is the STOP bit and the last one of the 11 bit sequence, the program must check the counter value. If the counter is equal to 8, that mean that the host has only read the 1st byte, or 16, that would mean that the mouse has read until the 2nd byte, it goes directly to the waiting state. If the counter is 24, the mouse has already sent the 3 bytes of data and the programs goes to the next case, STOP case.

```
             case STOP:

                  state = WAITING;
                  break;

             default:

                  break;
            }
      }
}
```

In the STOP case, the program reads the *flag_up* and the tick variables. When the clock edge falls and the *tick* value is 1, it initializes the variables *temp*, *tick* and reads the actual bit of data, that in this case it corresponds to the STOP bit. The program then goes back to the waiting state to start reading the data of the second and third bytes sent by the device.

As explained before, the final program requires the use of interruptions. They are configured as have been explained on section 4.5.1, but are implemented in parallel in a separate function named *configureInterrupts()* that looks like the following:

```
void configureInterrupts()
{

      printf("Configuring interrupts ... \n\r");
      fflush(stdout);
      while(1);

      /*......same code explained before.......*/

      return;
}
```

### 4.5.4 Implementation of the IP Mouse

Until now a complete pointing device system have been designed and implemented, but the aim of the project is to implement an IP Mouse, which means that the device must be able to communicate with the host through IP. This brings the system to include some Ethernet features, which need as well being implemented. At this section, the mouse code is used with some little modifications and improvements and there's also a sending interface part. The protocol used on the sending interface is UDP.

With the ColdFire the packet needs to be created from the beginning, adding the appropriate header and/or checksum to the packet body. This part has been adapted from the AN3470SW project [42] and demonstration code samples that can be found on the Freescale webpage. To create and send the packet, the program use several functions that can be looked at the code, since this section is only explaining the main functions used and that have been modified somehow.

Assuming that all the parameters have been already declared and created in other functions, the main functions used to send the device's data through the network, are int *emg_upsend* and *emg_send_data_via_udp*. The udpsend function sets the destination address, and port, the out buffer size and the length of the packets.

At the send_data_via_UDP function sends the data the user want to send. This function just takes the data received from the mouse and sends it through the port and to the destination previously set.

```
int emg_udpsend ( ip_addr dest_ip,
                  unsigned short dest_port,
                  char *outbuf,
                  int outlen)
{
      PACKET            pkt2;   // packet to send & free
      int               e;

      pkt2 = udp_alloc(outlen, 0);
      if(!pkt2)
            return -1;

      for( e=0; e<outlen; e++ )
            pkt2->nb_prot[e] = outbuf[e];

      pkt2->nb_plen     = outlen;
      pkt2->fhost       = dest_ip;
      pkt2->net         = NULL;
                                      // local port
      e = udp_send( dest_port, 0x1234,     pkt2 );

   if(e < 0)
   {
#ifdef NPDEBUG
      dprintf("tftp_udpsend(): udp_send() error %d\n", e);
#endif
      return e;
   }
   else
      return 0;
}
```

The emg_send_data_via_udp function is where the program put the variables that need to be sent.

```
//********************************************************************
void emg_send_data_via_udp(void)
{
      ip_addr           dest_ip = SERVER_IP;
      int         len;

      #if 1
            data_to_send[0] = Byte1;
            data_to_send[1] = Byte2;
            data_to_send[2] = Byte3;

      #endif

      // Send data_to_send to ip address det_ip, port PORT_NUMBER
if( emg_udpsend( dest_ip, PORT_NUMBER, (void *)data_to_send, len ) )
printf( "\nError sending via UDP " );
printf( "\nsent %d", len );
}
```

# 5 Data, Tests and Evaluation

## 5.1 IPM Interface

Previous chapters present the design, build and implementation of the IPM device. Some tests have already been made, but there's a need to test the complete system and check that it is working alright. This section shows the expected values and compares them with the obtained ones so one can see how the complete system is working.

To explain and compare the data, the PS/2 protocol is followed once again and table 5, from HID section needs to be reminded.

| | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|---|---|---|---|---|---|---|---|---|
| Byte 1 | Y overflow | X overflow | Y sign bit | X sign bit | Always 1 | Mid. Btn | Right Btn | Left Btn |
| Byte 2 | X movement | | | | | | | |
| Byte 3 | Y movement | | | | | | | |

Table 7. Movement Data Packet

As explained on the PS/2 protocol section, this table gives the appropriate information about the data that is sent from the mouse, and shows the data and information that can be found inside each byte sent.

Looking at the table and in general terms, when the mouse is sending button information, Byte 1 will carry that information and Bytes 2 and 3 should be 0, as the device is not doing any movement.

If the mouse is moving but no button is pressed, Byte 1 usually takes a value of '8', because bit 3 of Byte 1 is always set to 1. Even so, when no button is pressed but movement occurs, Byte 1 can take different values according to the movement made, as it also contains information of movement sign bit and movement overflow bit, which are set when the counters of the mouse go past 255.

Moreover if the movement made is on the X edge, Byte 2 will take any value from 0 to 255, and Byte 3 will be 0, as the movement is not made on the Y direction. By contrast, if the movement made is on the Y edge, Byte 2 will be 0 and Byte 3 can take any value from 0 to 255 according to the amount of movement recurred. As mentioned, if the movement achieved makes the mouse counters go over 255, Byte 1 can change its value because of the movement sign and the movement overflow bits.

Overall, data sent differs according to the event created: button pressed or X/Y movement. The comparison is made through a table that contains each frame received from the mouse, including the start bit, the 8 data bits, the parity bit and the stop bit.

Note that the order is upside down than the order on table 7. That is because the data is sent according to the Little endian format, which means that the Least Significative Bit (LSB) is sent first and the Most Significative bit (MSB) is the last bit sent.

The expected value table have some things in common independently of the event performed. Those things are the start bit, which is always 0, and the parity bit, which is always 1. The other values are written accordingly to the input event from the mouse. In order to make the results more visible, the most relevant bits or the ones more likely to change or to be checked for each case (input event) are highlighted in bold characters. Further details on this data is analysed and better detailed on the next lines.

If the LEFT BUTTON is pressed, the data expected and obtained are shown on the following table:

| | EXPECTED | | | | | | | | | | | OBTAINED | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Start | Bit0 | Bit1 | Bit2 | Bit3 | Bit4 | Bit5 | Bit6 | Bit7 | Parity | Stop | Start | Bit0 | Bit1 | Bit2 | Bit3 | Bit4 | Bit5 | Bit6 | Bit7 | Parity | Stop |
| Byte 1 | 0 | **1** | **0** | **0** | **1** | 0 | 0 | 0 | 0 | 1 | 1 | 0 | **1** | **0** | **0** | **1** | 0 | 0 | 0 | 0 | 1 | 1 |
| Byte 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| Byte 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |

Table 8. Left button expected vs. obtained values

Observe that in the obtained values from Table 8 the start bit is always 0 and the stop bit is always 1, as it was expected to be. The data sent on Byte 1 has a decimal value of ‘9’. Data from Bytes 2 and 3 is 0, as expected, because there is no movement made, just a button pressed. In that table, it is demonstrated that the parity bit is set when there are an even number of ones sent, so the final add up of 1’s at the frame is an odd number.

Visually, the data sent from the mouse when its left button has been pressed, has the shape shown on the next figures obtained with an oscilloscope:

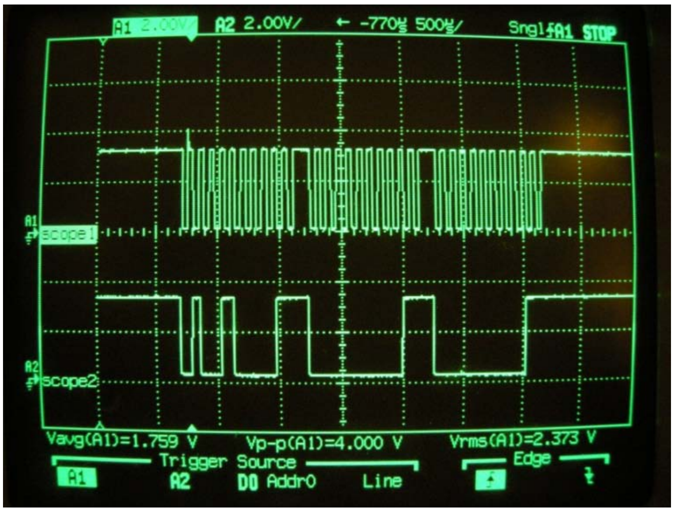


Figure 30. Three bytes sent from the mouse when Left Button is pressed

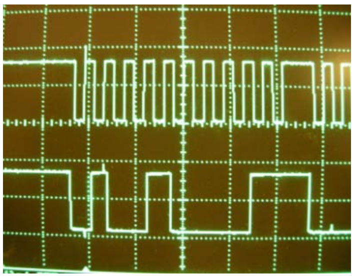


Figure 31. Left Button. First Byte

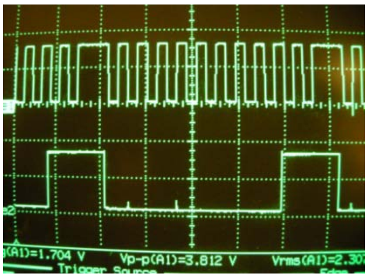


Figure 32. Left Button. Second Byte

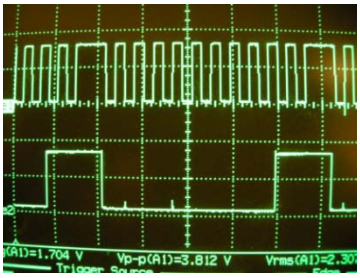


Figure 33. Left Button. Third Byte

Figures 30 to 33, contain the data sent from the mouse and extracted using an oscilloscope. Remembering that data is read on the falling edge of the clock, and as shown on previous table, the data (in binary) sent is:

Frame 1: 0 1001 0000 11. Frame 2: 0 0000 0000 11. Frame 3: 0 0000 0000 11.

Taking only the mouse information data (the 8 data bits alone without the start bit, parity bit and stop bit), the data sent corresponds with a decimal value of:

Byte1: 09 Byte2: 00 Byte3: 00

The data fulfils all the expectations.

Following with the analysis of the data, if the RIGHT BUTTON is pressed, the data expected and obtained is:

| | EXPECTED | | | | | | | | | | | OBTAINED | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Start | Bit0 | Bit1 | Bit2 | Bit3 | Bit4 | Bit5 | Bit6 | Bit7 | Parity | Stop | Start | Bit0 | Bit1 | Bit2 | Bit3 | Bit4 | Bit5 | Bit6 | Bit7 | Parity | Stop |
| Byte 1 | 0 | **0** | **1** | **0** | **1** | 0 | 0 | 0 | 0 | 1 | 1 | 0 | **0** | **1** | **0** | **1** | 0 | 0 | 0 | 0 | 1 | 1 |
| Byte 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| Byte 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |

Table 9. Right button expected vs. obtained values

As before, the start bit is always 0 and the stop bit is always 1. The data sent on Byte 1 has a decimal value of ‘10’. Data from Bytes 2 and 3 is 0, as expected, because no movement has occurred. The parity bit also corresponds to be an odd parity bit.

Extracting the data with the oscilloscope to see the real data transmitted, when the right button from the mouse is pressed, the frame values (in binary) obtained is:

Frame 1: 0 0101 0000 11. Frame 2: 0 0000 0000 11. Frame 3: 0 0000 0000 11.

Taking only the mouse information data (the 8 data bits alone without the start bit, parity bit and stop bit), the data sent corresponds with a decimal value of:

Byte1: 0a     Byte2: 00     Byte3: 00

Visually, the data sent from the mouse when its right button has been pressed, has the shape shown on the next figures:

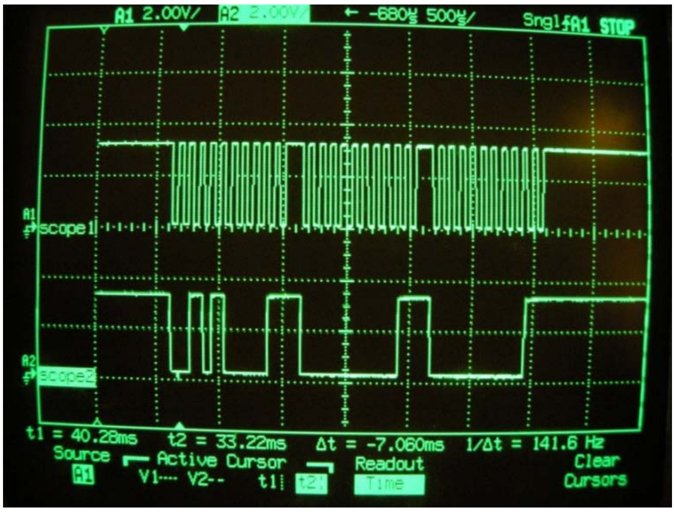


Figure 34. Three bytes sent from the mouse when right button is pressed

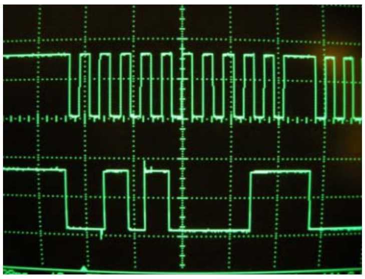


Figure 35. Right Button. First Byte

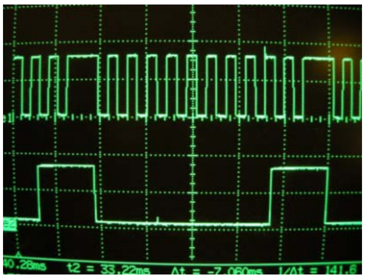


Figure 36. Right Button. Second Byte

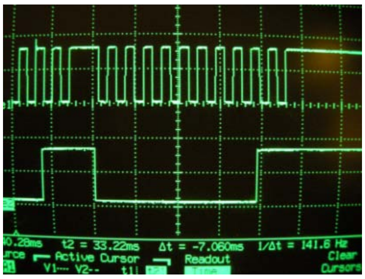


Figure 37. Right Button. Third Byte

IP Mouse only has two buttons implemented, so the next data analyzed to check if the mouse is working alright and sending the right data are the X and Y movements.

Moving the mouse toward the X edge to the right, the data expected and obtained are:

| | EXPECTED | | | | | | | | | | | OBTAINED | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Start | Bit0 | Bit1 | Bit2 | Bit3 | Bit4 | Bit5 | Bit6 | Bit7 | Parity | Stop | Start | Bit0 | Bit1 | Bit2 | Bit3 | Bit4 | Bit5 | Bit6 | Bit7 | Parity | Stop |
| Byte 1 | 0 | 0 | 0 | 0 | 1 | X | 0 | X | 0 | X | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| Byte 2 | 0 | X | X | X | X | X | X | X | X | X | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| Byte 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |

Table 10. X movement expected vs. obtained values

As on previous cases, the start bit is always 0 and the stop bit is always 1. In that case, the data sent on Byte 1 has a decimal value of ‘8’. But it could be also ‘24’ if the X sign bit (bit 4) was set, or ‘72’ if the X overflow bit (bit6) was set, or ‘88’ if both bits were set. Data from Bytes 2 takes a value of ‘1’ in this case, but it could be any valour from 1 to 255. Data from Byte 3 is 0, because the movement was made only on the X direction. The parity bit also corresponds to be an odd parity bit, it is ‘0’ on Byte 1, as there is only one bit set on the data byte sent, and it is ‘1’ on Bytes 2 and 3. Extracting the data with the oscilloscope to see the real data transmitted, the obtained values (in binary) are:

Frame 1: 0 0001 0000 11. Frame 2: 0 1000 0000 11. Frame 3: 0 0000 0000 11.

Following with what have been done on previous cases, taking only the mouse event information data, the data sent corresponds with a decimal value of:

Byte1: 08 Byte2: 01 Byte3: 00

Visually, the data sent from the mouse when its right button has been pressed, has the shape shown on the next figures:

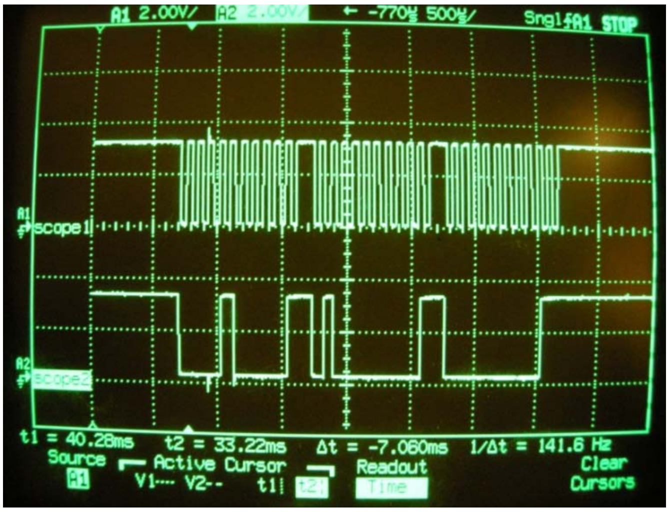


Figure 38. 3 bytes sent from the mouse when moved towards the X direction

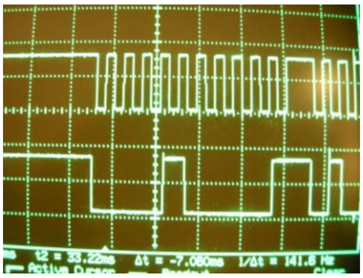


Figure 39. Right Movement First Byte

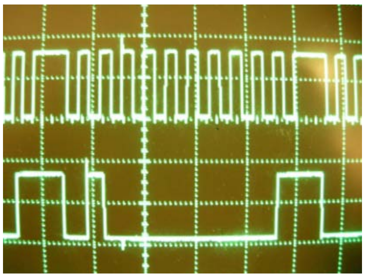


Figure 40. Right Movement Second Byte

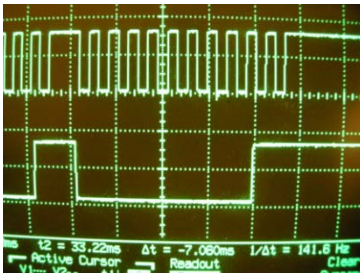


Figure 41. Right Movement Third Byte

Doing the same but with the mouse moving towards the X direction to the left, the data obtained is:

| | EXPECTED | | | | | | | | | | | OBTAINED | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Start | Bit0 | Bit1 | Bit2 | Bit3 | Bit4 | Bit5 | Bit6 | Bit7 | Parity | Stop | Start | Bit0 | Bit1 | Bit2 | Bit3 | Bit4 | Bit5 | Bit6 | Bit7 | Parity | Stop |
| Byte 1 | 0 | 0 | 0 | 0 | **1** | **X** | 0 | **X** | 0 | 0 | 1 | 0 | 0 | 0 | 0 | **1** | **1** | 0 | **0** | 0 | 0 | 1 |
| Byte 2 | 0 | **X** | **X** | **X** | **X** | **X** | **X** | **X** | **X** | X | 1 | 0 | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | 1 | 1 |
| Byte 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |

Table 11. Left movement (negative) expected vs. obtained values

As on the other data analyzed, the start bit is 0 and the stop bit is 1. In that case, the data sent on Byte 1 has a decimal value of '24', which means that the X sign bit is set, and data from byte 2 is '255'. Data from Byte 3 is '0', because the movement was made only on the X direction. The parity bit also corresponds to be an odd parity bit. Extracting the data with the oscilloscope to see the real data transmitted:

Frame 1: 0 0001 1000 11. Frame 2: 0 1111 1111 11. Frame 3: 0 0000 0000 11.

Taking only the eight data bits of information relevant for the event sent from the mouse, the data sent and received corresponds with a decimal value of:

Byte1: 24 Byte2: 255 Byte3: 00

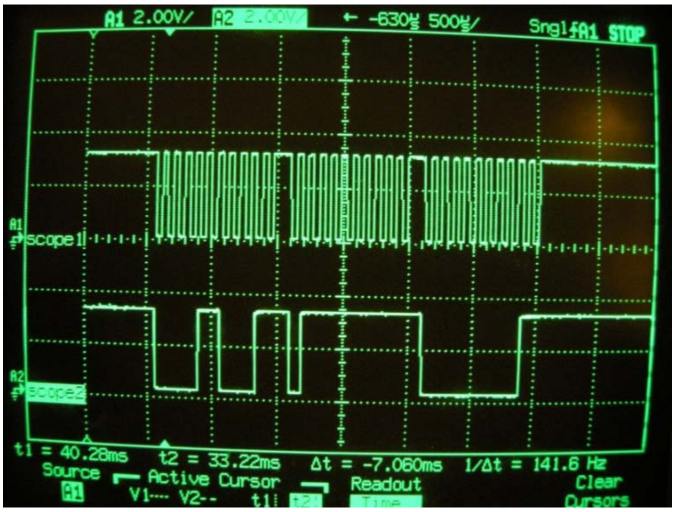


Figure 42. 3 bytes sent from the mouse when movement to the right has occurred

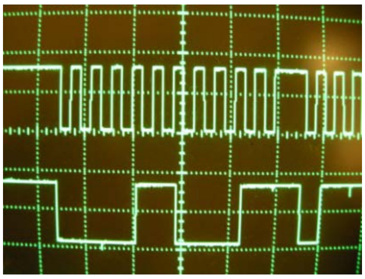


Figure 43. Right Movement. First Byte

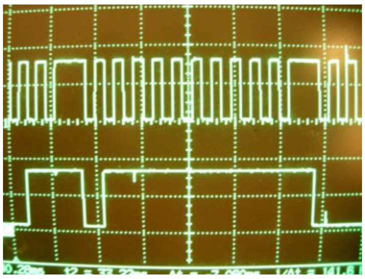


Figure 44. Right Movement. Second Byte

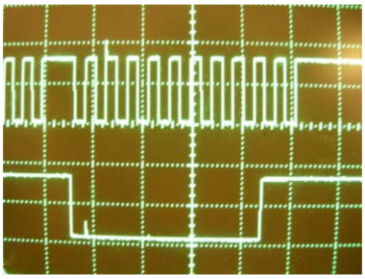


Figure 45. Right Movement. Third Byte

The Data from the mouse to be analyzed when moving the mouse towards the Y edge, up direction is:

| | EXPECTED | | | | | | | | | | | OBTAINED | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Start | Bit0 | Bit1 | Bit2 | Bit3 | Bit4 | Bit5 | Bit6 | Bit7 | Parity | Stop | Start | Bit0 | Bit1 | Bit2 | Bit3 | Bit4 | Bit5 | Bit6 | Bit7 | Parity | Stop |
| Byte 1 | 0 | 0 | 0 | 0 | **1** | 0 | **X** | 0 | **X** | **X** | 1 | 0 | 0 | 0 | 0 | **1** | 0 | **0** | 0 | **0** | **0** | 1 |
| Byte 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| Byte 3 | 0 | **X** | **X** | **X** | **X** | **X** | **X** | **X** | **X** | **X** | 1 | 0 | **0** | **1** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | 1 |

Table 12. Up movement expected vs. obtained values

As checked on all the other cases, start bit is '0' and stop bit is '1'. The data sent on Byte 1 has a decimal value of '8'. But it could be also '40' if the Y sign bit (bit 4) was set, '136' if the Y overflow bit (bit 6) was set, or '168' if both bits were set. Data from Byte 2 is '0' because the movement made is on the Y direction. Data from byte 3 is '2' in this case, though it could be any valour from '1' to '255' according with the distance moved. The parity bit also corresponds to be an odd parity bit.

Extracting the data with the oscilloscope, the real data transmitted obtained is:

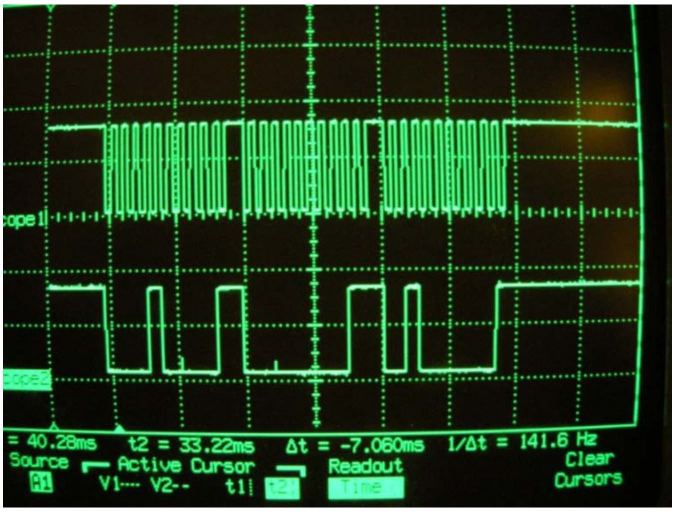


Figure 46. Up movement. 3 bytes sent from the mouse

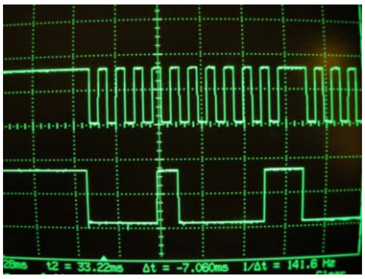


Figure 47. Up movement.
First Byte

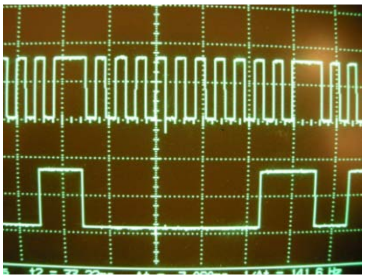


Figure 48. Up movement.
Second Byte

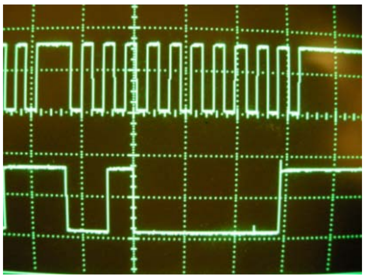


Figure 49. Up movement.
Third Byte

The movement data (in binary) sent and received in that case is:

Frame 1: 0 0001 0000 11. Frame 2: 0 0000 0000 11. Frame 3: 0 0100 0000 11.

Taking only the eight data bits of information relevant for the movement information sent from the mouse, the data sent and received corresponds with a decimal value of:

Byte1: 8 Byte2: 0 Byte3: 2

The data transmitted and received fulfils with all the expectations.

Performing the movement towards the Y edge but to the bottom direction, the data obtained is:

| | EXPECTED | | | | | | | | | | | OBTAINED | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Start | Bit0 | Bit1 | Bit2 | Bit3 | Bit4 | Bit5 | Bit6 | Bit7 | Parity | Stop | Start | Bit0 | Bit1 | Bit2 | Bit3 | Bit4 | Bit5 | Bit6 | Bit7 | Parity | Stop |
| Byte 1 | 0 | 0 | 0 | 0 | 1 | 0 | X | 0 | X | X | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| Byte 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| Byte 3 | 0 | X | X | X | X | X | X | X | X | X | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 |

Table 13. Up movement expected vs. obtained values

As presented on all the other cases, the start bit is 0 and the stop bit is 1. In this case, the data sent on Byte 1 has a decimal value of '40', that means that the Ysign bit has been set. Data from Byte 2 is '0' because the movement made is on the Y direction and Data from Byte 3 is '253' in this case, which remains inside the 1 to 255 range.

The parity bits are set according to the odd parity bit. Extracting the data transmitted with the oscilloscope, the results obtained are:

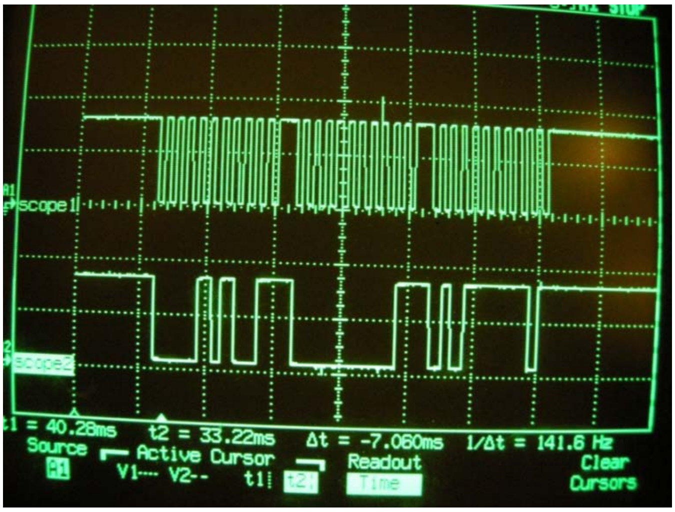


Figure 50. Down movement. 3 bytes sent from the mouse

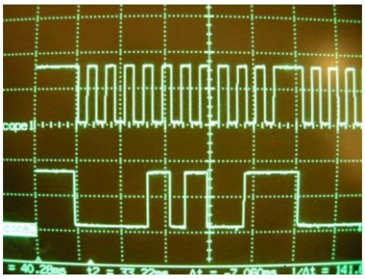


Figure 51. Down movement. First Byte

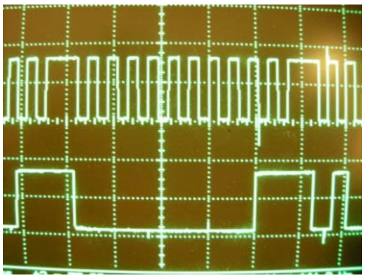


Figure 52. Down movement. Second Byte

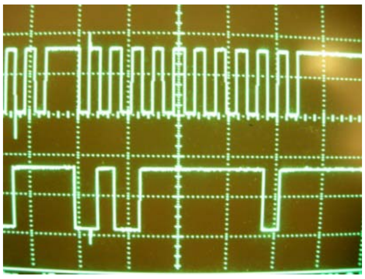


Figure 53. Down movement. Third Byte

The movement data (in binary) sent and received obtained on the oscilloscope (see figures 50-53) for this case are:

Frame 1: 0 0001 0000 11. Frame 2: 0 0000 0000 11. Frame 3: 0 1011 1111 11.

Taking only the eight data bits of information relevant for the movement information sent from the mouse, the data obtained have a decimal value of:

Byte1: 8 Byte2: 0 Byte3: 2

The data transmitted and received fulfils all the expectations.

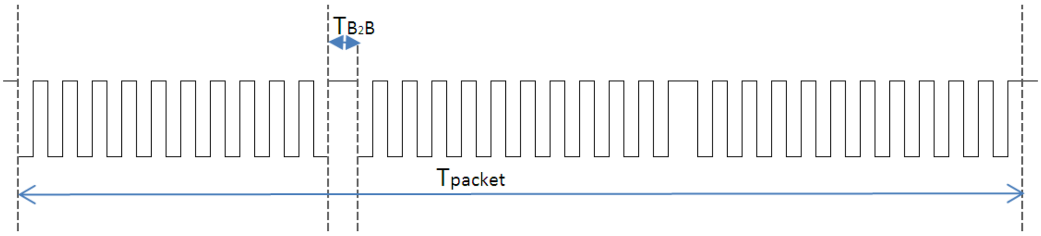
## 5.2 Timing and Frequency Specifications

The mouse works with a power supply of 5 V. Checking the data and clock lines with the oscilloscope, can be observed that both data and clock lines have a voltage of 3.4 V approximately.

The clock frequency obtained with the frequency measure option from the same oscilloscope, the IP Mouse is 11.76 kHz. Visually from the oscilloscope, the frequency is 10 kHz approximately. The difference obtained between the two values is because of the resolution of the oscilloscope screen. Anyways, the device frequency remains inside the range 10 – 16.7 kHz.

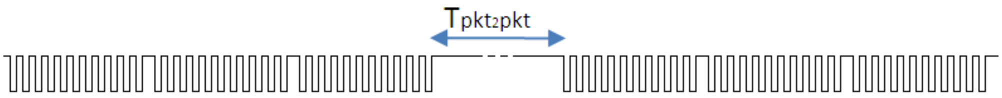
As a curiosity, it was observed that when the mouse sends its data bytes, it sends the 3 bytes in just 2975 µs.

The time from byte to byte is approximately 138µs and the time to send data packets consecutively is 7.060 ms. Timing diagrams are shown below:



$T_{B2B}$ = 138 µs
$T_{packet}$ = 2.975 ms

Figure 54. Timing of a byte sent from IPM



$T_{pkt2pkt}$ = 7.060 ms

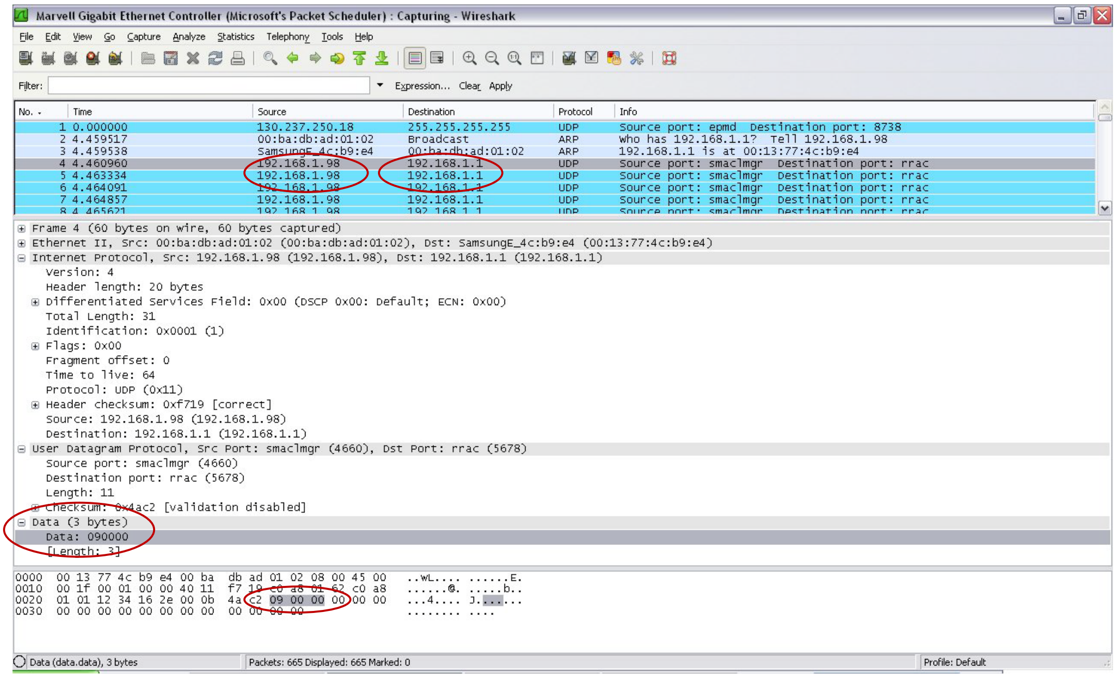
Figure 55.Timing between packets sent from IPM

## 5.3 Sending Packets Interface

The system is now proved to be working alright. The next trials were to see if the data was sent ok. To do so, one can either look at the watchdog window of the compiler, CodeWarrior software, or use any external program to capture packets and see what’s in them. In this case, WireShark [43] software has been used, as it is free software and easy to work with.

With the HyperTerminal one can only know if the packets are being sent or they have produced an error, so here comes a capture of the WireShark window to proof the board was sending the data packets alright. Some buttons and movements have been realized in order to get more than a packet.

WireShark gives all the information needed for the packets received. It shows the source and destination addresses, the protocol used for the packet, the source and destination ports used, and the number of packets sent with its length and data. Some examples of screen captures have been included. The information obtained when sending a left button packet is:


Marvell Gigabit Ethernet Controller (Microsoft's Packet Scheduler) : Capturing - Wireshark
File Edit View Go Capture Analyze Statistics Telephony Tools Help
Filter:
Expression... Clear Apply
Data (data.data), 3 bytes
Packets: 665 Displayed: 665 Marked: 0
Profile: Default

```
No.  Time        Source              Destination        Protocol  Info
  1 0.000000    130.237.250.18      255.255.255.255    UDP       Source port: epmd  Destination port: 8738
  2 4.459517    00:ba:db:ad:01:02   Broadcast          ARP       Who has 192.168.1.1?  Tell 192.168.1.98
  3 4.459538    SamsungE_4c:b9:e4   00:ba:db:ad:01:02  ARP       192.168.1.1 is at 00:13:77:4c:b9:e4
  4 4.460960    192.168.1.98        192.168.1.1        UDP       Source port: smaclmgr  Destination port: rrac
  5 4.463334    192.168.1.98        192.168.1.1        UDP       Source port: smaclmgr  Destination port: rrac
  6 4.464091    192.168.1.98        192.168.1.1        UDP       Source port: smaclmgr  Destination port: rrac
  7 4.464857    192.168.1.98        192.168.1.1        UDP       Source port: smaclmgr  Destination port: rrac
  8 4.465621    192.168.1.98        192.168.1.1        UDP       Source port: smaclmgr  Destination port: rrac
```

```
⊞ Frame 4 (60 bytes on wire, 60 bytes captured)
⊞ Ethernet II, Src: 00:ba:db:ad:01:02 (00:ba:db:ad:01:02), Dst: SamsungE_4c:b9:e4 (00:13:77:4c:b9:e4)
⊟ Internet Protocol, Src: 192.168.1.98 (192.168.1.98), Dst: 192.168.1.1 (192.168.1.1)
    Version: 4
    Header length: 20 bytes
  ⊞ Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
    Total Length: 31
    Identification: 0x0001 (1)
  ⊞ Flags: 0x00
    Fragment offset: 0
    Time to live: 64
    Protocol: UDP (0x11)
  ⊞ Header checksum: 0xf719 [correct]
    Source: 192.168.1.98 (192.168.1.98)
    Destination: 192.168.1.1 (192.168.1.1)
⊟ User Datagram Protocol, Src Port: smaclmgr (4660), Dst Port: rrac (5678)
    Source port: smaclmgr (4660)
    Destination port: rrac (5678)
    Length: 11
  ⊞ Checksum: 0x4ac2 [validation disabled]
⊟ Data (3 bytes)
    Data: 090000
    [Length: 3]
```

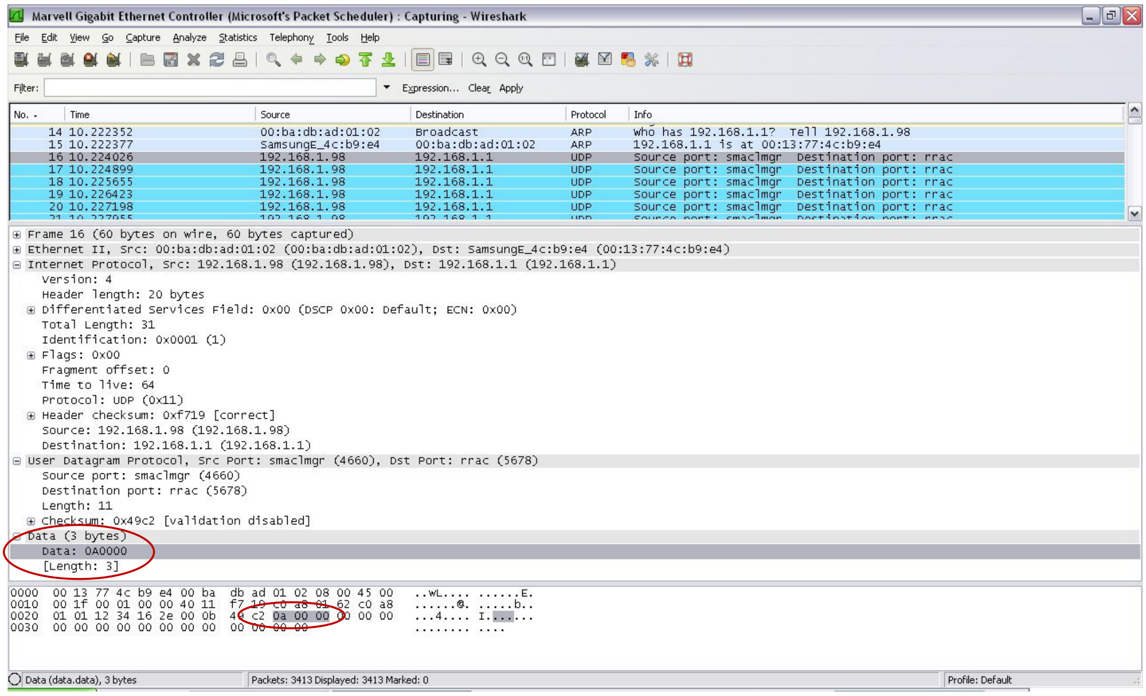
```
0000   00 13 77 4c b9 e4 00 ba  db ad 01 02 08 00 45 00   ..wL.... ......E.
0010   00 1f 00 01 00 00 40 11  f7 19 c0 a8 01 62 c0 a8   ......@. .....b..
0020   01 01 12 34 16 2e 00 0b  4a c2 09 00 00 00 00 00   ...4.... J.......
0030   00 00 00 00 00 00 00 00  00 00 00 00               ........ ....
```

Figure 56. WireShark screen capture for left button

As shown on the figure 56, the source address is the board IP, 192.168.1.98 and the destination address is the computer address, 192.168.1.1. The program also shows that the protocol used for the packet is UDP and that the source port is 4660 and the destination port used is 5678. Those port numbers could be other values, it all depends on how the code was programmed, and they were chosen randomly.

The source and destination addresses and ports are easy to check, as the values coincide, but the data sent from the mouse need to be more analyzed. The mouse sends data information in 3-byte packets, so the lengths of the packets received should be 3 bytes. The data information is different depending on the event sent or the movement occurred, as shown at all the examples from the previous section. In that case, according to the PS/2 protocol and to the Table 7, as a left button was pressed, the packet received should be of a decimal value 09 00 00 (according to the first, second and third Bytes). On the figure, there is this value marked with a red circle, showing that the data received is the data expected.

Looking the capture for the right button event:


Marvell Gigabit Ethernet Controller (Microsoft's Packet Scheduler) : Capturing - Wireshark
File Edit View Go Capture Analyze Statistics Telephony Tools Help
Filter:
Expression... Clear Apply

```
No. -  Time        Source              Destination        Protocol  Info
    14 10.222352   00:ba:db:ad:01:02   Broadcast          ARP       Who has 192.168.1.1?  Tell 192.168.1.98
    15 10.222377   SamsungE_4c:b9:e4   00:ba:db:ad:01:02  ARP       192.168.1.1 is at 00:13:77:4c:b9:e4
    16 10.224026   192.168.1.98        192.168.1.1        UDP       Source port: smaclmgr  Destination port: rrac
    17 10.224899   192.168.1.98        192.168.1.1        UDP       Source port: smaclmgr  Destination port: rrac
    18 10.225655   192.168.1.98        192.168.1.1        UDP       Source port: smaclmgr  Destination port: rrac
    19 10.226423   192.168.1.98        192.168.1.1        UDP       Source port: smaclmgr  Destination port: rrac
    20 10.227198   192.168.1.98        192.168.1.1        UDP       Source port: smaclmgr  Destination port: rrac
```

```
⊞ Frame 16 (60 bytes on wire, 60 bytes captured)
⊞ Ethernet II, Src: 00:ba:db:ad:01:02 (00:ba:db:ad:01:02), Dst: SamsungE_4c:b9:e4 (00:13:77:4c:b9:e4)
⊟ Internet Protocol, Src: 192.168.1.98 (192.168.1.98), Dst: 192.168.1.1 (192.168.1.1)
    Version: 4
    Header length: 20 bytes
  ⊞ Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
    Total Length: 31
    Identification: 0x0001 (1)
  ⊞ Flags: 0x00
    Fragment offset: 0
    Time to live: 64
    Protocol: UDP (0x11)
  ⊞ Header checksum: 0xf719 [correct]
    Source: 192.168.1.98 (192.168.1.98)
    Destination: 192.168.1.1 (192.168.1.1)
⊟ User Datagram Protocol, Src Port: smaclmgr (4660), Dst Port: rrac (5678)
    Source port: smaclmgr (4660)
    Destination port: rrac (5678)
    Length: 11
  ⊞ Checksum: 0x49c2 [validation disabled]
⊟ Data (3 bytes)
    Data: 0A0000
    [Length: 3]
```

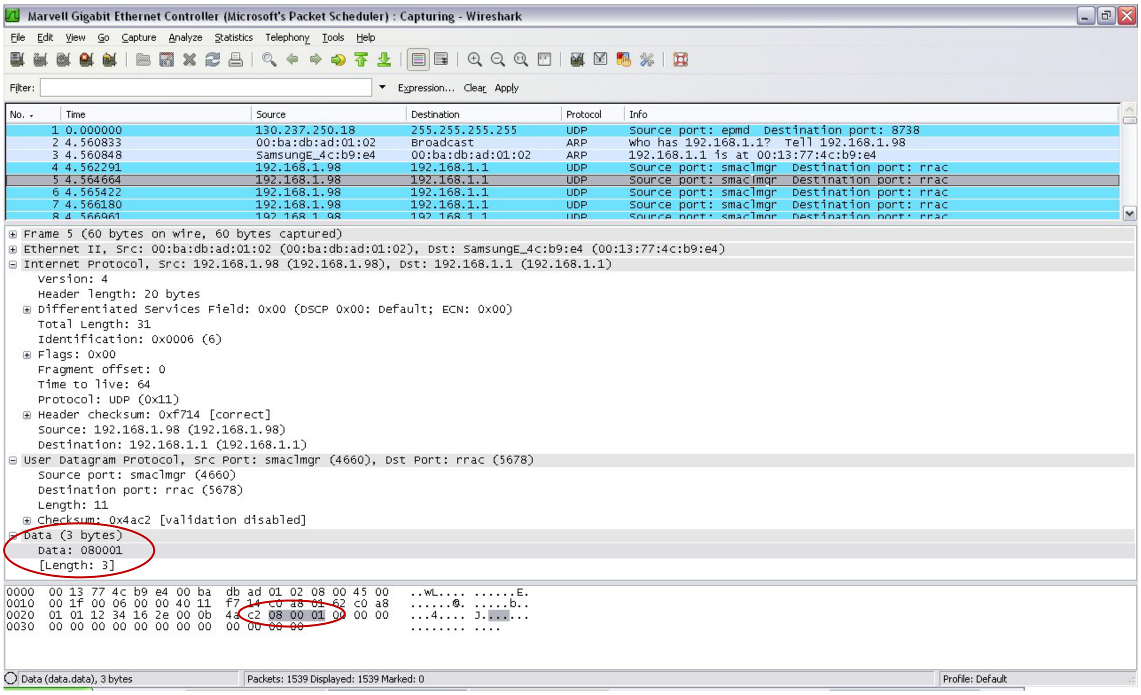
```
0000  00 13 77 4c b9 e4 00 ba  db ad 01 02 08 00 45 00   ..wL.... ......E.
0010  00 1f 00 01 00 00 40 11  f7 19 c0 a8 01 62 c0 a8   ......@. .....b..
0020  01 01 12 34 16 2e 00 0b  49 c2 0a 00 00 00 00 00   ...4.... I.......
0030  00 00 00 00 00 00 00 00  00 00 00 00               ........ ....
```

Data (data.data), 3 bytes
Packets: 3413 Displayed: 3413 Marked: 0
Profile: Default

Figure 57. WireShark screen capture for the right button

The addresses are still the same, but now, the expected data value (in decimal) received, as the right button has been pressed, is 0a 00 00. Once again, the result shows that the data received is the data expected. It has been red circled on the screen capture.

Finally, with a random movement performed, the screen print and the obtained results are:


Marvell Gigabit Ethernet Controller (Microsoft's Packet Scheduler) : Capturing - Wireshark
File Edit View Go Capture Analyze Statistics Telephony Tools Help
Filter:
Expression... Clear Apply

```
No. -   Time        Source               Destination          Protocol  Info
     1 0.000000     130.237.250.18       255.255.255.255      UDP       Source port: epmd  Destination port: 8738
     2 4.560833     00:ba:db:ad:01:02    Broadcast            ARP       Who has 192.168.1.1?  Tell 192.168.1.98
     3 4.560848     SamsungE_4c:b9:e4    00:ba:db:ad:01:02    ARP       192.168.1.1 is at 00:13:77:4c:b9:e4
     4 4.562291     192.168.1.98         192.168.1.1          UDP       Source port: smaclmgr  Destination port: rrac
     5 4.564664     192.168.1.98         192.168.1.1          UDP       Source port: smaclmgr  Destination port: rrac
     6 4.565422     192.168.1.98         192.168.1.1          UDP       Source port: smaclmgr  Destination port: rrac
     7 4.566180     192.168.1.98         192.168.1.1          UDP       Source port: smaclmgr  Destination port: rrac
     8 4.566961     192.168.1.98         192.168.1.1          UDP       Source port: smaclmgr  Destination port: rrac
```

```
⊞ Frame 5 (60 bytes on wire, 60 bytes captured)
⊞ Ethernet II, Src: 00:ba:db:ad:01:02 (00:ba:db:ad:01:02), Dst: SamsungE_4c:b9:e4 (00:13:77:4c:b9:e4)
⊟ Internet Protocol, Src: 192.168.1.98 (192.168.1.98), Dst: 192.168.1.1 (192.168.1.1)
    Version: 4
    Header length: 20 bytes
  ⊞ Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
    Total Length: 31
    Identification: 0x0006 (6)
  ⊞ Flags: 0x00
    Fragment offset: 0
    Time to live: 64
    Protocol: UDP (0x11)
  ⊞ Header checksum: 0xf714 [correct]
    Source: 192.168.1.98 (192.168.1.98)
    Destination: 192.168.1.1 (192.168.1.1)
⊟ User Datagram Protocol, Src Port: smaclmgr (4660), Dst Port: rrac (5678)
    Source port: smaclmgr (4660)
    Destination port: rrac (5678)
    Length: 11
  ⊞ Checksum: 0x4ac2 [validation disabled]
⊟ Data (3 bytes)
    Data: 080001
    [Length: 3]
```

```
0000  00 13 77 4c b9 e4 00 ba  db ad 01 02 08 00 45 00   ..wL.... ......E.
0010  00 1f 00 06 00 00 40 11  f7 14 c0 a8 01 62 c0 a8   ......@. .....b..
0020  01 01 12 34 16 2e 00 0b  4a c2 08 00 01 00 00 00   ...4.... J.......
0030  00 00 00 00 00 00 00 00  00 00 00 00               ........ ....
```

Data (data.data), 3 bytes
Packets: 1539 Displayed: 1539 Marked: 0
Profile: Default

Figure 58. Screen capture for movement

In that case, the mouse was moved very little towards the Y edge. The result should be, no button pressed for the first byte. The X movement byte (second byte) should take the value ‘0’, as the movement has been done to the Y direction, and the third byte (the one showing Y movement) should take some value, in that case, as it has been very few moved, should take a small value, like 1 or 2.

So the expected values for this movement are 08 00 0x. And the obtained values, as shown on the figure are 08 00 01, which confirms the hypothesis.

With all the tests made for the data sent and received, one can conclude that the IP Mouse is working as expected, and fulfils all the requirements expected.

## 5.4 IPM Limitations and Costs

The IPM system has been tested and it works fine. It fulfils the initial requirements: is able to perform 2D-motion and send the data through Ethernet. The data received is the expected. Even though, the device is not visually seen as a cursor on a computer display, as it should need the programming of a new graphic interface for the whole system, and that is included with the suggested improves that can be done in future prototypes.

As mentioned on previous chapters, this is only a prototype, and its cost, design and manufacture can differ depending on where is required to be included the device, computer mouse itself, PDA, mobile phone, etc., an overview of the total cost of the device have been noted.

Prices are not exact, because in most of electronic components, the prices changes depending on the quantity asked, as more components are asked, lower is the price per unit. Moreover, the prices have been searched on internet, and some of them are USA prices, so the values can differ from USA and Europe, and it also depends on the distributer or the store chosen. Even so, different prices have been compared and most common prices found or an average between them has been included on the next table.

Considering all the statements described, a general overview about the total cost of the IPM system is outlined on the next page:

| COMPONENT | DESCRIPTION | MANUFACTURER | PACKAGE | UNIT PRICE[2] | QUANT. | TOTAL PRICE |
|---|---|---|---|---|---|---|
| ColdFire M52233 | student Learning kit | Freescale | | 99 USD | 1 | 70 € |
| Copper-clad board | Double-sided<br>1.5mm/35µ Cu | | 100x160mm | < 5€ | 1 | 5€ |
| TPS61072 DDRC | IC Boost Converter<br>600mA SW | Texas Instruments | TSOT23-6 | 1.50 €/1 unit | 1 | 1.50 € |
| MAX3224 ECAP | RS-232 transceiver | MAXIM | 20-SSOP | 2.28 USD | 1 | 1.60 € |
| 74LCX86 | X-OR gate | Fairchild Semiconductors | SOP-14 | 0.671 USD<br>0.404 € | 1 | 0.5 € |
| BC546 | NPN transistor | | TO92 | 0.05€ | 2 | 0.1 € |
| CDPH4D19FNP-4R7MC | SMD Power Inductor<br>4.7µH±20% | Sumida | Surface Mount | 0.756 USD | 1 | 0.53 € |
| Capacitors | SMD capacitors<br>0.1 µA / 50 V | Murata | 0603 X7R | 0.0327 € | 7 | 0.23 |
| Capacitors | SMD capacitors<br>10 µA / 25V | Taiyo Yuden | 1206 X5R | 0.283 € | 1 | 0.28 |
| Resistors | SMD resistors<br>5% tolerance | Uni-Ohm | 0805 | 0.0018 € | 7 | 0.0126 |
| Total price | | | | | | 79.75€ |

Table 14. Overview of the prices of the different components used and final cost of the IPM prototype

With the research of the material and components used to build the IPM system, can be concluded that the complete system would cost a total amount of 80 Euro plus the price of a commercial PS/2 mouse device. That price is affordable, because the price of the components can be cheaper if they are bought in bigger quantities. Moreover, if the system may be intended to be built-in any electronic device, as a mobile phone or a PDA, the cost would be lower, because the whole system could be included in just a board, without the need of the complete evaluation kit from Freescale.

[2] As mentioned, all the prices are found on the website, and on different websites, that can be found on the Reference section numbered under the items [55] to [64 ].

# 6 Conclusion

In this work, an IP mouse prototype is presented. IPM is intended to provide multi cursor environment in a multi-application context, such as a windowing system. Even so, this work shows one single device capable of performing the usual 2 button and 2D motion pointer device. The IPM system performs that alright according to the PS/2 protocol and adds Ethernet features to send its data to a computer.

Based on technology and computer applications are growing up fast and that there are already some advanced systems like windowing systems taking advance and control of the screen and new ways of pointer devices are being investigated and actually arriving to the market, this prototype improved or something similar could become a common peripheral device for everybody in a not so far future.

The IPM system fulfils the previous desired characteristics and even being actually a limited function pointer device, it can be improved and be used as a multi pointer in future improved prototypes of the same. The use of the fully functioning IPM could facilitate the interaction with the same application in several scenarios like meetings, conferences, team work or school lectures.

Future improvements and functions to add to the system are countless, some of them are explained on the next subsection.

# 7 Future Work

Until now, the IP-Mouse has been prototyped, described and developed. IPM shows a new way to interact with a computer by having a pointer that can be connected through IP to any host compute.

The aim of the IPM is to provide multiple collaboration, which means, to allow different users to use different applications at the same time, or, to allow multiple cursors to interact with one application also at the same time.

The completed design shows a conventional mouse motion, two buttons and 2D motion. Knowing that, the immediate work that could be done, would be to extend the protocol and drivers in order to get a 3D pointer device (3D-IPM). This could be done by changing the code and following the PS/2 extended protocol. A 3D-IPM device could be helpful in future graphic interface systems such as in medicine or building design applications.

One may think that it can be a chaos to use multiple cursors on the same display. Therefore another improvement that could be done to the device in order to solve that problem is to research and find how to ID each device and to identify them on the screen. That could be done using the X Windowing server and toolkits. X Windowing has already some tools that focus the active window or event. Making the system compatible to X Windowing, could include some more improvements such as think how to distinguish cursors easily from one another. That could be to build each cursor of a different colour, to change the pointer shape, etc.

Future editions of IP-Mouse should study the possibility of using different protocols, such as USB, or even better, wireless communications, one may say Bluetooth, Zig-bee and especially 3G cellular. That could depend on the use that one want to give to the device, considering the distance and time of reaction of the same. That could help in multitudinary conferences, where everybody can take active part during the conference. Thinking of the same conference situation, it would be great if one device could connect at any system just identifying itself to its server or to the applications.

Applications for multi-pointing devices are endless. In my opinion, many of the existent software could be improved and modified in order to support IPM. But the more immediate things I think about when I imagine real nice working multi-cursor devices are computer games. Normally, the games for more than 1 or 2 players, the players have to wait for their turns. While one is playing the other players have to wait for their turn to come, and conform in seeing the other games. With multi-cursor, every player could be playing at the same time against the other, what could be, on my opinion, much more fun and competitive. The only thing that should also be improved is to increase the display size to make the game interface more comfortable.

Far from the games applications, IPM system could be introduced on educational software and applications. The TIC are rapidly increasing in schools and nowadays everybody is tired to hear politicians talking about if children on schools should have its own computer. In some countries (see Spain) there is already a test, where each child will have its own computer instead of the conventional school books. The intention is to innovate in resources and to approach the students to the technology. With multi-cursor, this could be even better, because everyone could interact actively with the common application that the teacher, in that case, could be explaining at the “blackboard” that, in this case would be a big display.

Moreover, in developing-world classrooms, where financial resources may be scarce, or even in developed countries, now that everybody is talking about the crisis, the addition of a few mice is far easier and cheaper than a PC per child. Apart from that would encourage team working, which has been demonstrated that is the best way to learn. Moreover, this would not only work in school, it could only take an important part in multitudinary conferences, meetings, etc.

More options can be analyzed in order to add features to the IP Mouse. During this work, multi-pointing features for having multi-user on one machine have been predominated, but looking further, there could be studied a way to provide IPM with memory and think how this memory could be used to cut and paste data from one computer to another, not just from one application to another. Depending on the memory size, the files memorized could be multimedia files, so the mouse could work as a temporary memory stick to copy files from one computer to another with ease and fast.

A new age of pointers is coming, and soon they will be devices of common use at work and free time applications.

# 8 References

[1] **World Wide Web Page**
GUIdebook>Articles> “The X-Window System” by Dick Pountain, from Byte, 1989. [On-line]. Accessed on August, 2009 at URL: http://guidebookgallery.org/articles/thexwindowsystem

[2] **World Wide Web Page**
The PS/2 Mouse/Keyboard Protocol by Adam Chapweske, 2003 [On-line]. Accessed on October, 2009 at URL: http://www.computer-engineering.org/ps2protocol/

[3] **World Wide Web Page**
68k/ColdFire. [On-line]. Accessed on May, 2009 at URL: http://www.freescale.com/webapp/sps/site/homepage.jsp?code=PC68KCF

[4] **World Wide Web Page**
M5223X Product Summary Page.[On-line]. Accessed on June, 2009 at URL: http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=MCF5223X&nodeId=0162468rH3YTLC00M95448

[5] **Technical Report, Patent**
Engelbart, D.C., (1970). *X-Y Position Indicator for Display System* (Pat. Number 3,541.541). Palo Alto, California. Assignor to Stanford Research Institute, Menlo Park, California. USA

[6] **Company**
International Business Machines Corporate (IBM Corporation). Headquarters:1 New Orchard Road. Armonk, New York 10504-1722. United States. Corporate URL: http://www.ibm.com/us/en/

**[7] Company**

Apple Inc. Headquarters: Infinite Loop, Cupertino, CA 95014.United States. Official corporate URL: http://www.apple.com

**[8] Company**

Microsoft Corporation. Address: Redmond, WA 98052-8300. United States. Official URL: http://www.microsoft.com

**[9] Company**

Agilent Technologies. Corporation Address: 5301 Stevens Creek Blvd, Santa Clara CA 95051. United States. Corporate URL: http://www.home.agilent.com/

**[10] Technical Report**

Stewart J., Bederson B., and Druin A. (1999) *Single Display Groupware: A model for Co-present Collaboration*. Conference on Human Factors in Computing Systems, CHI 1999, ACM Press

**[11] Technical Report**

Bier E.A, and Freeman S., (1991). *MMM: A User Interface Architecture for Shared Editors on a Single Screen*. Proceedings of the fourth annual ACM Symposium on User Interface Software and Technology

**[12] Technical Report**

**Wallace**, G., Bi, P., Li, K., and Anshus, O. (2004): *A Multi-Cursor X Window Manager Supporting Control Room Collaboration.* Princeton University, Computer Science, Report No. TR-0707-04, July 2004

**[13] Technical Report**

Hutterer, P., Close, B. S., and Thomas, B. H. (2006): *TIDL: Mixed Presence Groupware Support for Legacy and Custom Applications*. In *Proceedings of the seventh conference on Australasian user interfaces.* Hobart, Australia, 2006

**[14] Technical Report**

Peter Hutterer and Bruce H. Thomas. (2006) *Groupware Support in the Windowing System.* Australia, 2006. Project for the School of Computer and Information Science,University of South Australia. Mawson Lakes SA 5095

**[15] World Wide Web**

MPX: The Multi-Pointer X Server by Peter Hutterer. (2008) [On-line] Accessed on May, 2009 at URL: http://wearables.unisa.edu.au/mpx/

**[16] Technical Report**

Udai Singh Pawar, Joyojeet Pal and Kentaro Toyama. *Multiple Mice for Computers in Education in Developing Countries*.(2006)

**[17] Technical Report**

Tsutomu Kawai, Joutarou Akiyama and Minoru Okada.(2009): *Point-to-Multipoint Communication Protocol PTMP and Evaluation of Its Performance*. Department of Information Electronics, Graduate School of Engineering, Nagoya University Education Center for Information Processing, Nagoya University Furo-cho, Chikusa-ku, Nagoya, 464-01, Japan

**[18] World Wide Web**

What is a KVM Switch? [On-line] Accessed on December 2009 at URL: http://www.tech-faq.com/kvm-switch.shtml

**[19] World Wide Web**

Mobile Air Mouse by RPATech. [On-line] Accessed on November 2009 at the URL: http://www.mobileairmouse.com/

**[20] Company**

RPA Tech, a Boston based development company specializing in web, desktop and mobile software solutions for companies large and small. http://rpatechnology.com/

**[21] Computer software**

CodeWarrior IDE (Version 6.3) [Computer Software]. (2005) from URL: http://www.freescale.com/webapp/sps/site/overview.jsp?code=CW_SUITES&tid=CWH

**[22] Company**

Freescale Semiconductor. Headquarters: Austin, TX. United States. Corporate URL: http:// www.freescale.com

**[23] Datasheet**

*TPS61072 90% efficient synchronous boost converter with 600 mA switch.* (January 2007) by Texas instruments.

**[24] Company**

Texas instruments. Address: 12500 TI Boulevard, Dallas, Texas 75243, USA. Accessed on August 2009 at the URL: http://www.ti.com/

**[25] Datasheet**

*BCN546;BCN547. NPN General Purpose Transistors*. (1999) from Philips. Doc. order No.: 9397 750 05677. URL: http://www.semiconductors.philips.com

**[26] Datasheet**

*74LCX86: Low Voltage Quad 2-Input Exclusive-OR Gate with 5V Tolerant Inputs.* (March 1999) Fairchild semiconductor.

**[27] Datasheet**

*MAXIM. MAX3224ECAP. ±15kV ESD-Protected, 1µA, 1Mbps, 3.0V to 5.5V, RS-232 Transceivers with AutoShutdown Plus*. (1998). Doc No. 19-1339; Rev 0; 1/98.

**[28] Company**

MAXIM. Address: Maxim Integrated Products, Inc. 120 San Gabriel Drive Sunnyvale, CA 94086. USA**.** Corporate URL: http://www.maxim-ic.com/

**[29] Computer software**

EagleCAD (version 5.4.0) [Computer software]. 19620 Pines Blvd. Pembroke Pines, FL 33029. USA. Official URL: http://www.cadsoft.de/

**[30] Computer Software**

CircuitCAM. [computer software] from URL http://www.lpkf.com/products/rapid-pcb-prototyping/software/circuitcam-lite.htm

**[31] Computer Software**

Board Master. [computer software] At URL: http://www.lpkf.es/productos/creacion-rapida-prototipos-pcb/software/boardmaster.htm

**[32] Product**

LPKF protomat S42. LPKF Laser & Electronics AG. Addess: Osteriede 7, D-30827 Garbsen. Germany. Official URL: http://www.lpkfusa.com/protomat/s42.htm

**[33] World Wide Web**

Human Interface Devices.[On-line]. Accessed on June 2009 at URL: http://en.wikipedia.org/wiki/Human_interface_device

**[34] Computer software**

Windows [Computer Software] . Microsoft Corporate. Address: Redmond, WA 98052-8300. United States. Official URL: http://www.microsoft.com/WINDOWS/

**[35] Computer Software**

Mac OS X [Computer Software] . Cupertino, CA: Apple Computer Inc. Corporate URL: http://www.apple.com/macosx/

**[36] Computer software**

Palm OS [Computer Software]. Access, adquiring company for Palmsource Inc. Corporate headquarters: Hirata Bldg, 2-8-16 Sarugaku-cho, Chiyoda-ku, Tokyo 101-0064. Official URL: http://www.access-company.com/home.html

**[37] Institute of Technology**

Massachusetts Institute of Technology (MIT). Computer Science and Artificial Intelligence Laboratory. Accessed on October 2009. Address: MIT Computer Science and Artificial Intelligence Laboratory. The Stata Center, Building 32. 32 Vassar Street. Cambridge, MA 02139. USA. Institute URL: http://www.csail.mit.edu/

**[38] Publication**

Xlib – C Language X Interface. X Consortium Standard. X version 11. James Gettys and Robert W. Scheifler. Release 6.7 Draft, 2002. Accessed on July 2009 from the URL: http://www.xfree86.org/current/xlib.pdf

[39] **Technical Report**

Official Internet Protocol Standards (2007). [On-line] Accessed on December 2009 at URL: http://www.rfc-editor.org/rfcxx00.html

[40] **Publication**

*M52233DEMO. Demonstration Board for Freescale MCF52233*. (2006) Axiom Manufacturing. 2813 Industrial Lane, Garland, TX 75041. United States

[41] **Technical Report**

Freescale Semiconductor. M52233RM. *MCF52235 ColdFire® Integrated Microcontroller Reference Manual.* 2813 Industrial Lane, Garland, TX 75041. United States

[42] **Technical Report**

Freescale Semiconductor. *MCF52235 ColdFire® Integrated Microcontroller Reference Manual. Doc No. MCF52233RM.* Rev.5 09/2007.Headquarters: ARCO Tower 15F**,** 1-8-1, Shimo-Meguro, Meguro-ku, Tokyo 153-0064, Japan.

[43] **Computer Software**

WireShark [Free Computer Software]. Free Software Foundation, Inc. Headquarters: 59 Temple Place, Suite 330, Boston, MA 02111-1307 *USA* URL: http://www.wireshark.org/.

[44] **Technical Report**

Freescale Semiconductor. (2005) *ColdFire® Family Programmer's Reference Manual. Doc No. CFPRM. Rev.3 03/25.Headquarters:* ARCO Tower 15F**,** 1-8-1, Shimo-Meguro, Meguro-ku, Tokyo 153-0064, Japan.

[45] **World Wide Web**

What is the X Window System by O'Reilly.(2005) [ On-line] Accessed on June 2009 at URL http://linuxdevcenter.com/pub/a/linux/2005/08/25/whatisXwindow.html

[46] **World Wide Web**

PS/2 Mouse by Louis Ohland. [On-line] Accessed on June 2009 at URL http://www.tavi.co.uk/ps2pages/ohland/mouse.html#Signals

**[47] Technical Report**
Freescale Semiconductor. *MCF523x Integrated Microprocessor Hardware Specification* by Microcontroller Division. Doc. No. MCF5235EC, Rev. 2, 08/2006. Headquarters: ARCO Tower 15F**,** 1-8-1, Shimo-Meguro, Meguro-ku, Tokyo 153-0064, Japan.

**[48] Technical Report**
MCF52235 ColdFire Microcontroller Data Sheet. Doc. No. MCF5235DS, Rev. 7, 08/200. Headquarters: ARCO Tower 15F**,** 1-8-1, Shimo-Meguro, Meguro-ku, Tokyo 153-0064, Japan.

**[49] World Wide Web**
Eagle PCB -> LPKF Milling Machine Mini-How-To by Steve D. Sharples. (2002) [On-line] Accessed on March 2009 at URL http://optics.eee.nottingham.ac.uk/eagle/eagle2lpkf.html

**[50] World Wide Web**
ECE Illinois. Department of Electrical and Computer Engineering
University of Illinois at Urbana-Champaign [On-line] Accessed on March 2009 at URL http://www.ece.illinois.edu/eshop/pcbdesign/Programs.htm

**[51] World Wide Web**
Computer Mouse History Timeline by Z. Perry, 2009. [On-line] Accessed on December 2009 at URL http://hubpages.com/hub/computer-mouse-history

**[52] World Wide Web**
Computer Mouse – History – , 2004. [On-line] Accessed on December 2009 at URL http://www.youtube.com/watch?v=0MUOn_nJmRA

**[53] World Wide Web**
HowStuffWorks “How Computer Mice Work” by Marshall Brain and Carmen Carmack, 2008. [On-line] Accessed on December 2009 at the URL http://computer.howstuffworks.com/mouse3.htmhttp://www.youtube.com/watch?v=0MUOn_nJmRA

**[54] World Wide Web**

History of the computer mouse by Alex Vochin, 2009. [On-line] Accessed on December 2009 at URL http://gadgets.softpedia.com/news/History-of-the-Computer-Mouse-3938-01.html

**[55]** Coldfire M52233 Student Learning Kit Product Summary Page by Freescale semiconductor. [On-line]. Accessed on December 09 at URL http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=M52233SLK&tab=Buy_Parametric_Tab&nodeId=0162468rH38031&pspll=1&fromSearch=false

**[56]** TEXAS INSTRUMENTS|TPS61072DDCR|Linear Voltage Regulator IC| Newark.com [On-line]. Accessed on December 09 at URL http://www.newark.com/jsp/displayProduct.jsp?sku=30M1829&CMP=AFC-SF

**[57]** TPS61072DDCR by Mouser Electronics. [On-line]. Accessed on December 09 at URL:http://es.mouser.com/Search/Refine.aspx?Ntt=*TPS61072DDCR*&N=1323038&Ntx=mode%252bmatchall&Ns=P_SField&OriginalKeyword=TPS61072DDCR&Ntk=Mouser_Wildcards

**[58]** Voltage Regulators – Switching DC DC Converters & Controllers | Digi-Key by Digi Key Corporation [On-line]. Accessed on December 09 at URL http://search.digikey.com/scripts/DkSearch/dksus.dll?wt.mc_id=supplyframe_0709_buynow&site=us&mpart=TPS61072DDCR

**[59]** Digi-Key – MAX3224ECAP-ND (Manufacturer – MAX3224ECAP) by Digi-Key Corporation [On-line]. Accessed on December 09 at URL http://search.digikey.com/scripts/DkSearch/dksus.dll?wt.mc_id=supplyframe_0709_buynow&site=us&mpart=MAX3224ECAP

**[60]** Digi-Key – 74LCX86M-ND (Manufacturer – 74LCX86M) by Digi-Key Corporation [On-line]. Accessed on December 09 at URL http://search.digikey.com/scripts/DkSearch/dksus.dll?wt.mc_id=supplyframe_0709_buynow&site=us&mpart=74LCX86M

**[61]** Order 74LCX86M Parts Online, Download Datasheets, View Manufacturer Info at Avnet Express by Avnet Electronics Marketing. [On-line]. Accessed on December 09 at URL http://avnetexpress.avnet.com/store/em/EMController?langId=-1&storeId=500201&catalogId=500201&action=products&N=0&hrf=http://www.supplyframe.com/distributor/inventory/search/74lcx86&term=74LCX86M

**[62]** 74LCX86M by Mouser Electronics[On-line]. Accessed on December 09 at URL http://es.mouser.com/Search/Refine.aspx?Ntt=*74LCX86M*&N=1323038&Ntx=mode%252bmatchall&Ns=P_SField&OriginalKeyword=74LCX86M&Ntk=Mouser_Wildcards

**[63]** Welcome to ELFA – one of northern Europe's leading supplier of electronics. By ELFA. [On-line]. Accessed on December 09 at URL https://www1.elfa.se/elfa~eu_en/b2b/catalogstart.do?tab=catalog

**[64]** Fixed – Digi-Key by Digi-Key Corporation. [On-line]. Accessed on December 09 at URL:http://search.digikey.com/scripts/DkSearch/dksus.dll?wt.mc_id=supplyframe_0709_buynow&site=us&mpart=CDPH4D19FNP-4R7MC

**[65]** **MPX demo** video (2007) by Peter Hutterer. http://www.youtube.com/watch?v=0MUOn_nJmRA

[66] **Film** *"Pirates of Silicon Valley"(*1999). Director : MArtyn Burke. Is about the History of the foundation of Apple Computer Corporation and Microsoft Inc and the early days of its founders and the companies.

# APPENDIX 1 – Final Code

The code explained on the implementation part is the first version of the IP Mouse implemented that was working. The final code, is a little improved and some functions have been added and changed. The final code used is the following:

```
//*******************************************************
// File name :   IPM UDP client.c
// Author:           Cristina Riera
//
// Part of this code was modified and adapted from a
// free Freescale sample written by Eric Gregory
//
// Description :     UDP client
//***********************************************************
#include "ipport.h"
#include "tcpapp.h"
#include "msring.h"
#include "menu.h"
#include OSPORT_H
#include "ipport.h"
#include "libport.h"
#include "udp.h"
#include <stdio.h>

#define  PORT_NUMBER             5678
#define  TEST_BUFFER             3
#define  INTER_PACKET_DELAY      1
#define  SERVER_IP               0xC0A80101 // 192.168.1.1

#define READ_CLOCK()((MCF_EPORT_EPPDR0 & MCF_EPORT_EPPDR_EPPD7)>>7)
#define READ_DATA()((MCF_GPIO_SETAN & MCF_GPIO_SETAN_SETAN4)>>4)

#define RESET_COMMAND 0xff
#define DATA_REPORTING_COMMAND 0xf4
#define SET_STREAM_MODE    0xea

void configurePins(void);
void configureInterrupts(void);
void readData(void);
void sendData(unsigned char Mdata);
unsigned char parity(unsigned char par);

#include "mcf52233_gpio.h"

volatile unsigned char flag_up = 0;
volatile unsigned char pinClock=0;
volatile unsigned char pinData=0;
volatile unsigned char data = 0;

volatile unsigned char tick = 0;
volatile unsigned char Data = 0;
volatile unsigned char Mparity = 0;
volatile unsigned char countB = 0;
unsigned char cnt = 0;
```

```
   enum _d2h
   {
         WAITING,
         ST_BIT,
         D0,
         D1,
         D2,
         D3,
         D4,
         D5,
         D6,
         D7,
         PARITY,
         STOP,
   };

typedef enum _d2h d2hState;

d2hState state;

   volatile unsigned char Byte1 = 0;
   volatile unsigned char Byte2 = 0;
   volatile unsigned char Byte3 = 0;

   int sth = 0;

//*****************************************************************
// Declare Task Object
//*****************************************************************

TK_OBJECT(to_freescale);
TK_ENTRY(tk_freescale);
struct inet_taskinfo freescale_task = {

   &to_freescale,
                                                          "FreeScale
Task",

   tk_freescale,

   NET_PRIORITY,
                                                          0x800
                                                    };

volatile unsigned char     data_to_send[TEST_BUFFER];


//*******************************************************************
// Declare a Socket structure and communications queue "msring"
//*******************************************************************
```

```
//***********************************************************
//
// emg_udpsend based on tftp_udpsend
// Written by Eric Gregori
//
// Send outbuf to dest_port at ip address dest_ip
//
//***********************************************************
int emg_udpsend (    ip_addr dest_ip,
                     unsigned short dest_port,
                     char *outbuf,
                     int outlen
                 )
{
   PACKET      pkt2;   // packet to send & free
   int         e;

   pkt2 = udp_alloc(outlen, 0);
   if(!pkt2)
         return -1;

   for( e=0; e<outlen; e++ )
         pkt2->nb_prot[e] = outbuf[e];

         pkt2->nb_plen     = outlen;
   pkt2->fhost      = dest_ip;
         pkt2->net         = NULL;
                                 // local port
         e = udp_send( dest_port, 0x1234,     pkt2 );

   if(e < 0)
   {
#ifdef NPDEBUG
      dprintf("tftp_udpsend(): udp_send() error %d\n", e);
#endif
      return e;
   }
   else
      return 0;
}

//************************************************************
//
// Sample function for a UDP client
//
//************************************************************
void emg_send_data_via_udp()
{
   ip_addr           dest_ip = SERVER_IP;
   int         len;

   if (!sth)
   {

   configureInterrupts();        // configuring my Interrupts

   printf("Writing!!!! \n");

   configurePins();
```

```
    // ------------------ HOST TO DEVICE--------------------

    flag_up = READ_CLOCK();

    sendData(RESET_COMMAND);  //Set the default values of the mouse

    sendData(SET_STREAM_MODE);

    sendData(DATA_REPORTING_COMMAND);   //Enables Data Reporting
                                        //from the mouse

    state = WAITING;

 // ------------------ DEVICE TO HOST ---------------------

    readData();
    }
          else if(sth)
          {
                readData();
          }

 }// end of void emg_send_data_via_udp()


 //***************************************************************
 //
 // UDP client init
 //
 //***************************************************************

 void emg_init_udp_client( void )
 {
 /* unsigned int      i;

    for( i=0; i<TEST_BUFFER; i++ )
          data_to_send[i] = i; */
 }

 //***************************************************************
 //
 // UDP client init
 //
 //***************************************************************

 void emg_udp_client_cleanup( void )
 {

 }
```

```
//**************************************************************
// The application thread works on a "controlled polling" basis:
// it wakes up periodically and polls for work.
//
// The task could aternativly be set up to use blocking sockets,
// in which case the loops below would only call the "xxx_check()"
// routines - suspending would be handled by the TCP code.
//
//
// FUNCTION: tk_emg_http_srv
//
// PARAM1: n/a
//
// RETURNS: n/a
//
//*************************************************************

TK_ENTRY(tk_freescale)
{
   while (!iniche_net_ready)
         TK_SLEEP(1);

   emg_init_udp_client();

   for (;;)
   {


         emg_send_data_via_udp();
         tk_yield();

         if (net_system_exit)
         break;
         }
         TK_RETURN_OK();

}

//*************************************************************
// create_freescale_task() - Written by Eric Gregori
// //  modified by Cristina Riera
//
// Free application provided for Freescale
//
// Insert the FreeScale task into the RTOS.
//*************************************************************
void create_freescale_task( void )
{
   int e = 0;

         e = TK_NEWTASK(&freescale_task);
         if (e != 0)
         {
      dprintf("Freescale task create error\n");
      panic("create_apptasks");
         }
}
```

```
void configurePins(void)
{

   MCF_GPIO_PANPAR = 0x00;      // Configuration of PORTAN as GPIO.

   MCF_GPIO_DDRAN = 0xC0;       // Configuration of pins AN7 and
AN6 as an OUTPUT

   MCF_GPIO_PORTAN = 0x00;      //Initialize M_clock and M_data
both high
}


void configureInterrupts()
{

// printf("Configuring interrupts ... \n\r");
// fflush(stdout);
// while(1);

         /* Enable IRQ signals on the port */
    MCF_GPIO_PNQPAR = 0
        | MCF_GPIO_PNQPAR_IRQ7_IRQ7;


    /* Set EPORT to look for both edges */
   MCF_EPORT_EPPAR0 = 0
        | MCF_EPORT_EPPAR_EPPA7_BOTH;

    /* Clear any currently triggered events on the EPORT  */
    MCF_EPORT_EPIER0 = 0
        | MCF_EPORT_EPIER_EPIE7;

    MCF_EPORT_EPFR0 |= 0x80;    /** Reset flag

    /* Enable interrupts in the interrupt controller */
   MCF_INTC0_IMRL &=  ~(MCF_INTC_IMRL_MASK7); //

   /* EPORT Data Direction Register */

   MCF_EPORT_EPDDR0 = 0x00;     // All pins INPUT

   return;
}
```

```
/******************************************************************
/
// This function reads the data bits from the mouse
// calculates the parity bit using the parity function
// and also sends the stop bit by pulling data high
//
// FUNCTION: sendData()
//
//----------------- HOST TO DEVICE COMMUNICATION -----------------
//
// PARAM1: n/a
//
// RETURNS: n/a
/****************************************************************/

void sendData(unsigned char Mdata)
{
   int i=0;
   int Mdata2 = 0;

   unsigned char par = parity(Mdata);

// flag_up = READ_CLOCK();

   MCF_GPIO_PORTAN = MCF_GPIO_PORTAN
                         | MCF_GPIO_SETAN_SETAN7;
   //Pull clock low

   cpu_pause(100);         //delay

   MCF_GPIO_PORTAN = MCF_GPIO_PORTAN
                         | MCF_GPIO_SETAN_SETAN6;      //Pull data
low while holding clock low


   MCF_GPIO_PORTAN = MCF_GPIO_PORTAN & 0x7F;      //Release clock


   while(flag_up) ;
   while(!flag_up) ;       // start bit

   while(i<8)
   {
         while(flag_up) ;  // do nothing
                     //the host sends the data while clock is high
         if(!flag_up)
         {
         Mdata2 = Mdata & 0x01;
         //set data line
         if (Mdata2) MCF_GPIO_PORTAN = MCF_GPIO_PORTAN & 0xBF;
         //reset data line
         else MCF_GPIO_PORTAN = MCF_GPIO_PORTAN | 0x40;

         Mdata = Mdata>>1;
         i++;

         }
         while(!flag_up) ;
   }
```

```
while (flag_up) ;

         if(par)     MCF_GPIO_PORTAN = MCF_GPIO_PORTAN & 0xBF;
         else        MCF_GPIO_PORTAN = MCF_GPIO_PORTAN | 0x40;
   // write parity bit

   while (flag_up) ;

         MCF_GPIO_PORTAN = MCF_GPIO_PORTAN & 0xBF; // stop bit

}
unsigned char parity(unsigned char Mdata)
{
   unsigned char par = 0;
   int p, aux=0;
   for (p=0;p<8;p++)
   {
         aux = Mdata & 0x01;
         par += aux;
         par &= 0x01;
         Mdata>>1;
   }
   par = ~par;
   return par;
}



/*******************************************************************
/
// This function reads the data bits from the mouse
// calculates the parity bit using the parity function
// and also sends the stop bit by pulling data high
// send the data via Ethernet
//
// FUNCTION: readData()
//
// PARAM1: n/a
//
// RETURNS: n/a
/******************************************************************/

void readData()
{
   ip_addr           dest_ip = SERVER_IP;
   int         len;
   int   sent = 0;

   len = TEST_BUFFER;
```

```
// ------------------ DEVICE TO HOST ---------------------

   state = WAITING;

   while(!sent)
   {
         switch(state)
         {
               case WAITING:

                     pinData = READ_DATA();
                     if(!flag_up)
                           if(!READ_DATA())  //START bit
                     {
                           Mparity = 0;
                           Data = 0;
                           tick = 0;
                           state = ST_BIT;
                     }
                     break;



               case ST_BIT:

                     if(flag_up && !tick)tick=1;
                     if(!flag_up && tick)
                     {
                           volatile unsigned char temp = 0;
                           tick = 0;
                           temp = READ_DATA();           //read D0
                           Data += temp;
                           countB++;
                           state = D0;
                     }

                     break;


               case D0:

                     if(flag_up && !tick)tick=1;
                     if(!flag_up && tick)
                     {
                           volatile unsigned char temp = 0;
                           tick = 0;
                           temp = READ_DATA();     //read D1
                           temp = temp << 1;
                           Data += temp;
                           countB++;
                           state = D1;
                     }

                     break;
```

```
case D1:

      if(flag_up && !tick)tick=1;
      if(!flag_up && tick)
      {
            volatile unsigned char temp = 0;
            tick = 0;
            temp = READ_DATA();     //read D2
            temp = temp << 2;
            Data += temp;
            countB++;
            state = D2;
      }

      break;


case D2:

      if(flag_up && !tick)tick=1;
      if(!flag_up && tick)
      {
            volatile unsigned char temp = 0;
            tick = 0;
            temp = READ_DATA();     //read D3
            temp = temp << 3;
            Data += temp;
            countB++;
            state = D3;
      }

      break;


case D3:

      if(flag_up && !tick)tick=1;
      if(!flag_up && tick)
      {
            volatile unsigned char temp = 0;
            tick = 0;
            temp = READ_DATA();     //read D4
            temp = temp << 4;
            Data += temp;
            countB++;
            state = D4;
      }

      break;
```

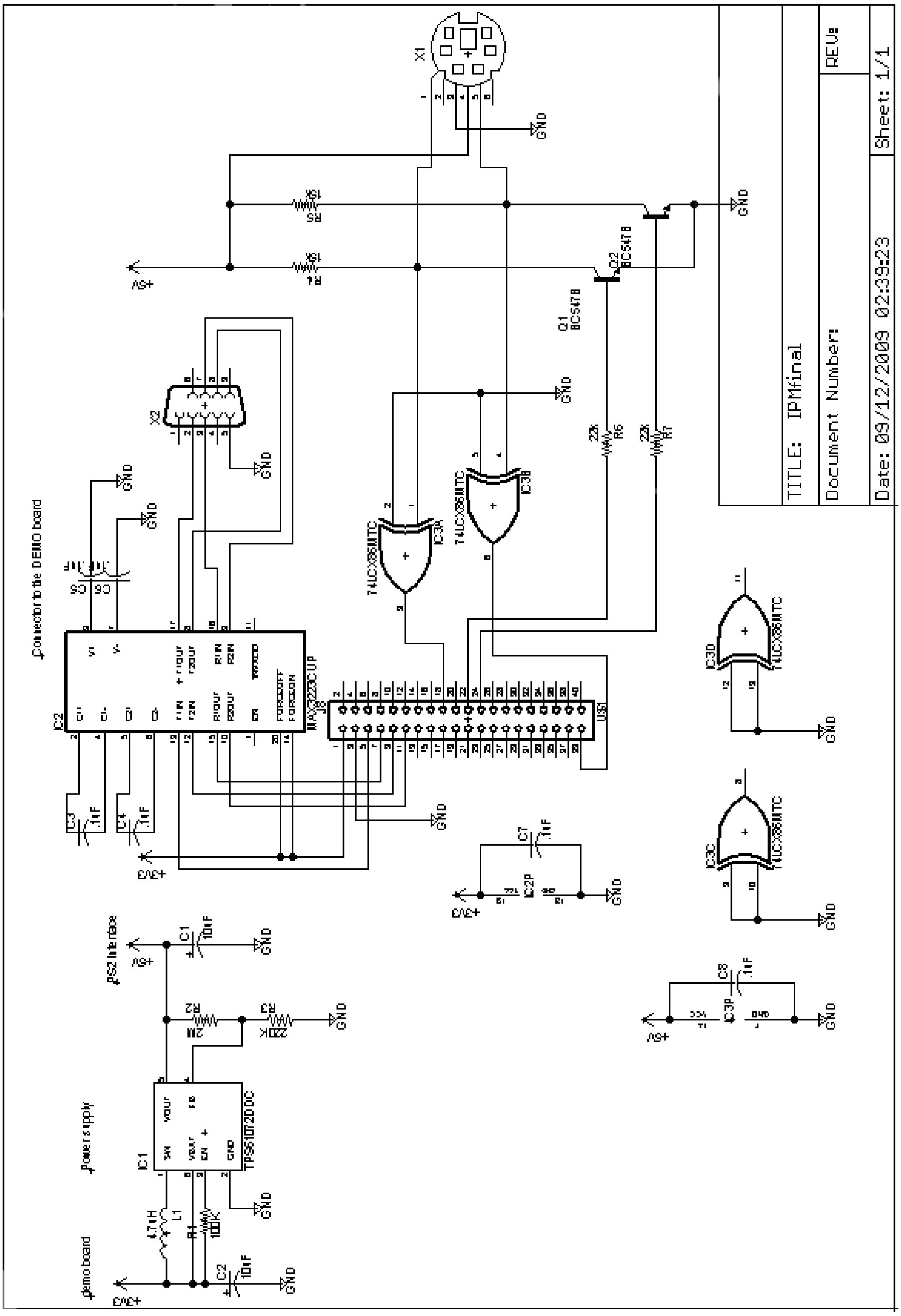
```
case D4:

      if(flag_up && !tick)tick=1;
      if(!flag_up && tick)
      {
            volatile unsigned char temp = 0;
            tick = 0;
            temp = READ_DATA();     //read D5
            temp = temp << 5;
            Data += temp;
            countB++;
            state = D5;
      }

      break;


case D5:

      if(flag_up && !tick)tick=1;
      if(!flag_up && tick)
      {
            volatile unsigned char temp = 0;
            tick = 0;
            temp = READ_DATA();     //read D6
            temp = temp << 6;
            Data += temp;
            countB++;
            state = D6;
      }

      break;


case D6:

      if(flag_up && !tick)tick=1;
      if(!flag_up && tick)
      {
            volatile unsigned char temp = 0;
            tick = 0;
            temp = READ_DATA();     //read D7
            temp = temp << 7;
            Data += temp;
            countB++;
            state = D7;
      }

      break;
```

```
case D7:

      if(flag_up && !tick)tick=1;
      if(!flag_up && tick)
      {
            volatile unsigned char temp = 0;
            tick = 0;
            temp = READ_DATA();     //read parity
            Mparity += temp;
            state = PARITY;
      }

      break;


case PARITY:

      if(flag_up && !tick)tick=1;
      if(!flag_up && tick)
      {
            volatile unsigned char temp = 0;
            tick = 0;

            temp = READ_DATA();     //read STOP bit

            if ((countB==8) && (cnt == 0))
            {
                  countB =0;
                  cnt++;
                  state = WAITING;
            }
            if((countB == 8) && (cnt != 0))
            {
                  Byte1 = Data;
                  state = WAITING;
            }
            if (countB==16)
            {
                  Byte2 = Data;
                  state = WAITING;
            }
            if (countB==24)
            {
                  Byte3 = Data;
                  countB = 0;
                  state = STOP;
            }
      }

      break;
```

```
                case STOP:
                      data_to_send[0] = Byte1;
                      data_to_send[1] = Byte2;
                      data_to_send[2] = Byte3;

          // Send data_to_send to ip address dest_ip,port PORT_NUMBER
                if( emg_udpsend  (  dest_ip,
                                    PORT_NUMBER,
                                    (void *)data_to_send,
                                    len )
                                  )
                      printf( "\nError sending via UDP " );
                      printf( "\nsent %d", len );

                      sth =1;
                      sent = 1;
          break;
          }

    }

 }
```

# APPENDIX 2 – Final Designs

## FINAL SCHEMATIC DESIGN



## FINAL PCB DESIGN